Universidad de las Ciencias Informáticas

Facultad 6

Título: Propuesta de arquitectura para desarrollar Sistemas de Información Geográfica sobre dispositivos móviles basado en el framework Quantum GIS.

Trabajo para optar por el Título de Ingeniero en Ciencias Informáticas

Autor: Guillermo Ortiz Alfonso

Tutor: Ing. Lisandra Escalona Griff

"Junio, 2012"

"Por norma, los sistemas de software no funcionan bien hasta que han sido utilizados y han fallado repetidamente en entornos reales"

Dave Parnas

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 6 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo el presente a los ____ días del mes de _____del año 2012.

_____ _____

Autor: Guillermo Ortiz Alfonso Tutor: Ing. Lisandra Escalona Griff

Datos de Contacto

Síntesis del Autor: Guillermo Ortiz Alfonso

Correo Electrónico: gortiz@estudiantes.uci.cu

Teléfono: 58272440

Síntesis de la Tutora: Ing. Lisandra Escalona Griff

Profesión: Ingeniera en Ciencias Informáticas

Años de graduada: 2 años

Correo Electrónico: lgriff@uci.cu

Teléfono: 8373102

Dedicatoria

Hoy culmino mis estudios universitarios, con la satisfacción de haberme graduado como Ingeniero en Ciencias Informáticas. Quisiera dedicar este gran triunfo a:

Mis padres, Guillermo M. Ortiz Peña y Rosa A. Alfonso Aldama, que han sido mis mejores amigos y me han dado su ejemplo y apoyo en todo momento. No hay palabra que llegue a manifestar el amor de los padres, pero yo puedo decir que nunca me ha faltado.

Mi hija Elizabeht I. Ortiz Menéndez, la luz de mis ojos, el angelito por el que me levanto cada día.

Mi novia Yenisleidys C. Miranda Linares, por darme su amor, su apoyo y no dejarme caer en los momentos difíciles, mi pensamiento eres tu Caridad.

Sinceramente

Guillermo Ortiz Alfonso

Agradecimientos

En este momento me vienen a la mente muchas personas a las que les debo agradecer por estar hoy aquí:

Primeramente a mis padres, Guillermo y Rosa por darme la vida, su amor y ejemplo, mi hermana Lisbeth, que siempre me protegió y apoyó, a mi abuela Ana, que aún hoy con mi tamaño y los años que tengo me mima, a mi novia Yenis, por soportar mis estados de ánimos y apoyarme siempre, a mi tío Carlos, que sus consejos siempre me han sido útiles, a mi prima Yunia, por levantarme el ánimo siempre con su manera tan peculiar.

De manera general quisiera agradecerles a todos mis familiares, tíos, primos, abuelos que de una forma u otra en estos años de vida me dieran algún consejo que me hiciera tomar el camino correcto y no alejarme de mis metas.

A la familia de Yenis, que me acogieron como un miembro más y me dieron su apoyo en todo momento, algo que no olvidaré nunca.

A mis amigos, que me han acompañado en estos años de estudios. Especialmente a los que en la universidad formamos el grupo que llamamos ANK, cuyo significado le revelaré en otro momento, cuando nuestros sueños se hagan realidad.

También quisiera agradecerles a mi tutora y profesores que me apoyaron en todo este tiempo.

Muchas gracias a todos, de corazón.

Sinceramente

Guillermo Ortiz Alfonso

Resumen

Los Sistemas de Información Geográfica son hoy día una tecnología de gran utilidad en el ámbito empresarial. Constituyen un instrumento de apoyo a la toma de decisiones, además de brindar un sinnúmero de funcionalidades como son la navegación sobre el mapa y la geolocalización de objetivos. Debido a las ventajas que poseen estos sistemas, su desarrollo y evolución han permitido que se puedan aplicar a todo tipo de dispositivos, desde los potentes ordenadores hasta los dispositivos móviles. El proyecto Control de Flotas que pertenece al centro GEYSED en la facultad 6 de la Universidad de las Ciencias Informáticas, se dedica a la construcción de Sistemas de Información Geográfica para estos dispositivos, donde ha alcanzado resultados tangibles en la tecnología Web, pero no posee una arquitectura que permita el proceso de desarrollo de este tipo de sistemas en tecnología de escritorio. Teniendo en cuenta que para el desarrollo de cualquier sistema informático es de vital importancia realizar previamente un adecuado diseño arquitectónico del mismo, se hace necesario realizar una propuesta de arquitectura para desarrollar Sistemas de Información Geográfica de escritorio sobre dispositivos móviles, lo cual representa la problemática a resolver del presente Trabajo de Diploma. La propuesta del diseño arquitectónico está basada en el uso de estilos y patrones que se han empleado con anterioridad en el desarrollo de dichos sistemas. Además, está en correspondencia con la soberanía tecnológica por la que aboga hoy Cuba, proponiendo que se utilicen herramientas y tecnologías libres para su desarrollo.

Palabras Claves: arquitectura, dispositivos móviles, Sistemas de Información Geográfica

Índice

Introducción

En la actualidad, con el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) han surgido nuevas herramientas que han aumentado la calidad y los resultados en la economía, la política y la sociedad. Estas aplicaciones permiten ampliar los beneficios de las esferas donde coexistan diferentes disciplinas; ejemplo de estas son los Sistemas de Información Geográfica (SIG) donde se encuentran integradas la Geografía, la Cartografía y la Informática.

Los SIG tienen una gran aceptación en la sociedad por la amplia gama de aplicaciones militares, económicas y sociales que brindan a los usuarios que interactúan con ellos. A nivel internacional no tienen una definición única que sea aceptada por todos los especialistas, pero para el desarrollo de esta investigación se utilizará el concepto que se muestra a continuación.

"Los Sistemas de Información Geográfica se pueden catalogar como la integración organizada de *hardware*, *software*, datos geográficos y personal, diseñado para capturar, almacenar, manipular, analizar y desplegar la información geográficamente referenciada" **(Gianfelici, 2008).** Estos sistemas permiten el análisis y la visualización de mapas en un entorno digital, con el fin de resolver problemas complejos de planificación y gestión, facilitando así la toma de decisiones.

En la última década se ha generalizado el uso de los SIG por la mayoría de las empresas del mundo debido a las ventajas que brindan estas herramientas como la localización de objetos georeferenciados o el cálculo de distancia. Muchas empresas e instituciones dedicadas a la producción de *software* han decidido destinar sus fondos al desarrollo de estas aplicaciones por su amplia comercialización, como es la compañía ESRI, que desarrolló una serie de Sistemas Información Geográfica llamados ArcGis.

El desarrollo de la tecnología actual ha permitido que los dispositivos móviles puedan realizar más funcionalidades. Estos en ocasiones han llegado a sustituir a las computadoras personales por su portabilidad. La evolución del teléfono móvil ha permitido disminuir su tamaño y peso, lo que permite comunicarse desde casi cualquier lugar. Aunque su principal función es la transmisión de voz, como en el teléfono convencional, su rápido desarrollo ha incorporado otras funciones como son cámara fotográfica, agenda, acceso a Internet, reproducción de vídeo e incluso GPS **(GUERRERO, 2010)**, además pueden incorporar aplicaciones más complejas como los Sistemas de Información Geográfica.

Cuba ha avanzado en el desarrollo de estos sistemas. Algunas empresas y centros universitarios relacionados con la producción de *software,* como son Geocuba y la Universidad de las Ciencias

Informáticas (UCI), que unificaron sus recursos para desarrollar GeneSig, una plataforma que permite el desarrollo de estas aplicaciones. En la universidad mencionada fue creado el proyecto Control de Flotas para la investigación y futuro desarrollo de estas aplicaciones en los dispositivos móviles, que pertenece al Centro de Geoinformática y Señales Digitales (GEYSED).

En proyecto antes mencionado se presentan dos líneas de investigación: el desarrollo de SIG para dispositivos móviles en la web, área en que se ha alcanzado resultados tangibles, pues se exhibe el producto MovilMap. La otra dirección de investigación tiene como objetivo el desarrollo de estos sistemas en tecnología de escritorio que permitan la consulta y análisis de la información geográfica desde cualquier lugar, además de la edición de los mapas. En estos momentos el proyecto no cuenta con una organización estructural para el desarrollo de estas aplicaciones. No se tienen identificados los componentes necesarios para la construcción de los Sistemas de Información Geográfica de escritorio en los dispositivos móviles. La documentación que existe sobre estas herramientas en el proyecto brinda poca información o nula, esto puede provocar que no se pueda reutilizar el código de la solución para el futuro. Estos factores influyen en que la ejecución de la aplicación no se termine en el tiempo planificado y que el costo aumente.

Dada la situación antes expuesta se identifica como **problema a resolver:** ¿Cómo definir la arquitectura de un Sistema de Información Geográfica de escritorio para dispositivos móviles?

Debido a esto se determina como **objeto de estudio:** El proceso de diseño de una arquitectura de *software* para el desarrollo de Sistemas de Información Geográfica de escritorio y como **campo de acción:** El diseño y descripción de la arquitectura de *software* para el desarrollo de Sistemas de Información Geográfica de escritorio para dispositivos móviles. Por lo que se define como **objetivo general:** Diseñar una arquitectura de un SIG de escritorio para dispositivos móviles.

Para lograr el objetivo planteado se presenta como **idea a defender:** Si se diseña una arquitectura de *software* para un SIG de escritorio en dispositivos móviles, entonces se obtendrá la línea base para el proceso de desarrollo de Sistemas de Información Geográfica en tecnología de escritorio para dispositivos móviles.

Con el fin de alcanzar el objetivo planteado se propuso desarrollar las tareas investigativas que se plantean a continuación:

1. Fundamentación de las tendencias actuales, tecnologías y conceptos más importantes relacionados con la arquitectura de los Sistemas de Información Geográfica.
2. Selección de los estilos y patrones de la arquitectura de *software* idóneos para el desarrollo de este tipo de aplicación.
3. Elaboración de la propuesta de línea arquitectónica a utilizar en el desarrollo del sistema.
4. Validación del diseño arquitectónico de la solución.

Después de haber realizado las tareas antes mencionadas se obtuvieron como resultados:

1. El diseño de la arquitectura para desarrollar Sistemas de Información Geográfica de escritorio para dispositivos móviles sobre la herramienta Quantum GIS.
2. La documentación generada a partir de la metodología de desarrollo de *software* utilizada.

Para desarrollar todas las tareas propuestas en dicha investigación, dentro de los métodos científicos existentes, fue necesario auxiliarse de los métodos teóricos Analítico-Sintético, Histórico-lógico y la Modelación. A continuación se describe como fueron aplicados en la investigación.

Analítico-Sintético

Permitió analizar y aumentar los conocimientos entorno al objeto de estudio a partir de consultar la bibliografía científica correspondiente, para después, haciendo uso de la síntesis, resumir y exponer los resultados obtenidos del análisis.

Histórico-lógico

Este método permitió realizar un análisis relacionado con las diferentes arquitecturas existentes de los principales Sistemas de Información Geográfica para dispositivos móviles. También posibilitó analizar los distintos patrones y estilos arquitectónicos, con el objetivo de encontrar los que más se ajustaron al desarrollo de los SIG para estos dispositivos.

Modelación

Este método permitió realizar un correcto modelado de la arquitectura de los SIG de escritorio para dispositivos móviles.

Resumen por capítulos

Capítulo I: Fundamentación Teórica: En este capítulo se hace un estudio del arte de la arquitectura de *software* y de los Sistemas de Información Geográfica, así como de los principales Sistemas Operativos (SO) usados en los dispositivos móviles. Se describen algunos conceptos fundamentales de la arquitectura. Además, se detallan y analizan las herramientas y tecnologías que se utilizan en el proceso de desarrollo de un SIG, se conforma la propuesta de tecnologías y herramientas a utilizar, y se exponen los elementos que fungieron como criterios de selección a la hora de elegir entre todas las opciones posibles. Conocimiento con el cual ya se puede realizar la descripción de la arquitectura.

Capítulo II: Descripción de la Arquitectura. En este capítulo se muestran los atributos de calidad que debe cumplir la propuesta de arquitectura realizada. Se evidencia la organización del contenido que tendrá la aplicación. Se realiza la descripción de la arquitectura teniendo en cuenta las 4+1 vistas que propone Kruchten. Luego de realizar la propuesta solo que evaluarla e identificar los posibles riesgos.

Capítulo III: Evaluación de la Arquitectura. En este capítulo se evalúa la arquitectura propuesta a partir del método de evaluación y la técnica seleccionada, que fueron utilizados en la validación de la arquitectura definida, resaltando la importancia que tiene el proceso de evaluación.

Capítulo I

Fundamentación Teórica

En el presente capítulo se realiza el análisis de algunos de los principales Sistemas de Información Geográfica existentes, con lo cual se pudo determinar las funcionalidades básicas de este tipo de sistemas e identificar la arquitectura que presenta cada uno de ellos. También se hace un estudio de las plataformas y los SO que se utilizan en estos dispositivos, pudiendo establecer una comparación entre ellos. Se identifican los patrones y estilos arquitectónicos, patrones de diseño, lenguajes de programación, herramientas de modelado y demás tecnologías que se pudieran emplear durante el desarrollo de un SIG.

1.1 Conceptos asociados al dominio de la investigación

Una vez definido el objetivo que se persigue con el desarrollo de esta investigación, surgen interrogantes que son necesarias responder para lograr un mejor dominio de los temas que se abordan.

1. **¿Qué se entiende por *software* libre?**

*S*ignifica que el *software* respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el software. Con estas libertades, los usuarios (tanto individualmente como en forma colectiva) controlan el programa y lo que hace. **(Free Software Foundation, 2001)**

Estas libertades han sido aceptadas en el mundo del desarrollo de *software*, pues brindan la posibilidad al desarrollador de adaptar un sistema y optimizarlo según sus necesidades. En los dispositivos móviles el *software* libre ha tenido un gran impacto producto a las características propias de su *hardware* y al entorno donde se desarrollan. Los programas de código abierto permiten una mayor flexibilidad de adaptar estos dispositivos a las necesidades de los usuarios.

2. **¿Qué son los dispositivos móviles?**

Estos dispositivos, conocidos como computadora de mano, *"Palmtop*[1]*"* o simplemente *handheld*[2], son de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o

[1] ***Palmtop**:* es un pequeño ordenador que literalmente cabe en la palma de la mano. Son prácticos para determinadas funciones, tales como guías telefónicas y calendarios.

intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales **(Dispositivos Móviles, 2010)**.

El continuo desarrollo de estos dispositivos, los han convertido en herramientas de uso cotidiano en la sociedad. Por lo que se han incorporado diferentes aplicaciones a los mismos. Algunos hoy cuentan ya hasta con Sistemas Operativos que les permite elevar las funcionalidades que brindan.

1.2 Sistemas Operativos de los dispositivos móviles y Sistemas de Información Geográfica

1.2.1 Plataformas y sistemas operativos en los dispositivos móviles

El sistema operativo es el programa o *software* más importante de un ordenador. Para que funcionen los otros programas, cada ordenador de uso general debe tener un sistema operativo. Los sistemas operativos realizan tareas básicas, tales como reconocimiento de la conexión del teclado, enviar la información a la pantalla, no perder de vista archivos y directorios en el disco, y controlar los dispositivos periféricos tales como impresoras y escáner. **(Sistema y recursos para tener éxito en internet, 2012)**

En la actualidad existe una gran diversidad en cuanto a los dispositivos móviles y a los Sistemas Operativos (SO) que estos utilizan. El SO de un celular es el programa base con el cual funciona el teléfono. Si se comparan con las computadoras, se estaría hablando de Windows, Mac OS o Linux. Se ha querido dar a conocer las características fundamentales de algunos de los SO que más se utilizan en los dispositivos móviles.

En la Tabla 1 se realiza una comparación entre los SO predominantes en los dispositivos móviles atendiendo a las principales características que son de interés para la arquitectura de *software*. Para conocer más información sobre estas particularidades de estos dispositivos, ver ***Anexo 1***.

[2] ***Handheld***: es una terminología inglesa, su significado es computadora de mano. Es también conocida como Asistentes Personales Digitales.

Sistema Operativo	Plataforma	Lenguaje del SDK/NDK	Dispositivos	Arquitectura	% de Popularidad
Symbian	Symbian	C++ Python	Smartphone	Arquitectura en Capas	65
Android	Google	SDK: java Python NDK: C++	PDA, Smartphone	Basada en Componentes	20.3
IOS	Apple	C,objective c	IPhone	Arquitectura en Capas	16.3
Windows Mobile	Microsoft	C++	Sistemas empotrados	Modular	7.6

Tabla 1. Plataformas y Sistemas operativos de los dispositivos móviles.

De la variedad de sistemas operativos que existen para estos teléfonos se escogen los SO Symbian y Android, pues además de ser los más distribuidos en el mundo, permiten el desarrollo de aplicaciones nativas para dispositivos móviles. Las aplicaciones nativas, son aquellas que se instalan en el dispositivo y se desarrollan con un lenguaje de programación del propio dispositivo. Este tipo de aplicaciones permiten una disminución del consumo de memoria en los móviles. Estos sistemas operativos permiten el desarrollo en los lenguajes C++, Java y Python.

Los dispositivos móviles, según sus funcionalidades se pueden categorizar como:

- ✓ **Dispositivo móvil de datos limitados:** tienen una pantalla pequeña, principalmente basada en pantalla de tipo texto con servicios de datos generalmente limitados a mensajería de texto y comunicación mediante el protocolo de acceso inalámbrico WAP[3]. Un típico ejemplo de este tipo de dispositivos son los teléfonos móviles convencionales.
- ✓ **Dispositivo móvil de datos básicos:** tienen una pantalla de mediano tamaño, (entre 120 x 120 y 240 x 240 píxeles), menú o navegación basada en íconos por medio de una "rueda" o cursor, y que ofrecen correo electrónico, lista de direcciones, mensajería de texto, y un navegador web básico. Un ejemplo de este tipo de dispositivos son los BlackBerry y los teléfonos inteligentes.
- ✓ **Dispositivo móvil de datos mejorados:** tienen pantallas de medianas a grandes (por encima de los 240 x 120 píxeles), navegación de tipo *stylus*, y que ofrecen las mismas características que el dispositivo móvil de datos básicos más aplicaciones nativas como

[3] **WAP:** *Wireless Application Protocol*, según sus siglas en inglés (protocolo de aplicaciones inalámbricas).

aplicaciones de *Microsoft Office Mobile* (*Word, Excel, Power Point*) y aplicaciones corporativas usuales, en versión móvil y portales intranet.

Para la propuesta de arquitectura que se pretende realizar, se considera que se deben tener en cuenta los Teléfonos Inteligentes, que según sus funcionalidades se encuentran en la tercera categoría explicada anteriormente. Los *Smartphone*, según su nombre en inglés, son teléfonos construidos sobre una plataforma móvil; tienen la capacidad de usarse como una computadora de bolsillo, llegando incluso a remplazar a una computadora personal en algunos casos. El desarrollo continuo de estos dispositivos, ha provocado que existan cada vez aplicaciones complejas destinadas a ellos. Estas exigen capacidad de memoria y de procesamiento, como son los Sistemas de Información Geográfica, en tecnología Web o escritorio.

1.2.2 Soluciones existentes de Sistemas de Información Geográfica

De los múltiples SIG que existen en la actualidad, se ha escogido una muestra de los más significativos, con el fin de realizar una comparación en cuanto a las características que presentan y que son de interés para la investigación. Dicha comparación permitió conocer si existen elementos en común para la descripción de una arquitectura y funcionalidades que sean propias de los Sistemas de Información Geográfica en los dispositivos móviles. Ver Tabla 2.

En la selección del SIG a tomar como referencia para el desarrollo de la propuesta de arquitectura, se tuvo en cuenta en primer lugar que el *software* estuviera desarrollado en tecnología libre. Además se consideró que fuera de escritorio para dar solución a la problemática planteada en la introducción de esta investigación. Dada la diversidad de SO existentes en el mercado, es necesario profundizar en el estudio de la arquitectura de un SIG que sea multiplataforma, es por ello que este es un elemento a tener en cuenta para la selección.

SIG	Sistema Operativo	Lenguaje	Formatos Soportados	Tecnología	Arquitectura	Licencia
GvSIG Mobile	Android	Java	Vectoriales y ráster	Escritorio	Multihilo	GPL[4]
GvSIG Mini	Android	Java	Vectoriales y ráster	Web	Multihilo	GPL
ArcGIS Mobile	Windows Mobile, Vista, XP	C++	Vectoriales y ráster	Escritorio	Por capas y orientada a servicios	Propietario
Quantum GIS	Windows Xp, Windows 7, Ubuntu	C++ y Python	Vectoriales y ráster	Escritorio	No tiene una definida	GPL v2

Tabla 2. Características de los Sistemas de Información Geográfica.

Se puede notar que la construcción de estos sistemas desde el punto de vista de la arquitectura es muy heterogénea, pues existen muy pocas coincidencias en los estilos arquitectónicos aplicados en su desarrollo. Como se observa en la tabla anterior los SIG que reúnen las características necesarias para la selección son GvSIG Mobile y Quantum GIS. Estos presentan algunas diferencias que se deben conocer a la hora de seleccionar una u otra tecnología. Se puede decir que las funcionalidades que estos sistemas presentan son las propias de un SIG: la navegación y el análisis del terreno por mencionar algunas, aunque también tienen algunas avanzadas como es el trabajo con GPS[5]. Las particularidades de estas herramientas para los dispositivos móviles se encuentran por lo general en la interfaz de usuario que brindan.

Para lograr el objetivo propuesto se selecciona la herramienta Quantum GIS. Esta posibilita el consumo de Servicios Web de Mapas (WMS, según sus siglas en inglés), soportando los estándares de la *Open Geospatial Consortium* (OGC), este sistema está respaldado por una comunidad de desarrolladores y se utiliza como base dentro de la UCI para el desarrollo de SIG en tecnología de escritorio. Además, Quantum GIS por la forma en que está programado permite la reutilización de su código, característica a tener en cuenta para un futuro desarrollo. Es una aplicación extensible mediante la incorporación de complementos, lo que posibilita agregar nuevas funcionalidades sin necesidad de realizar cambios profundos en el código. Es un potente editor de datos geográficos, lo que permite desarrollar un trabajo más eficiente en el terreno.

[4] **GPL:** por sus siglas en inglés, en español Licencia Pública General.
[5] **GPS:** Sistema de Posicionamiento Global.

Seleccionada ya le herramienta SIG que se utilizará como base para el desarrollo de la investigación. Se necesita analizar qué elementos de la arquitectura son necesarios para realizar una buena propuesta y obtener los resultados esperados.

1.3 Elementos de la Arquitectura de Software

Al revisar la literatura especializada sobre la Arquitectura de *Software* (AS), se pueden encontrar numerosas definiciones para el tema, cada una con planteamientos diversos. En general dichos planteamientos lejos de contradecirse, más bien se complementan en cuanto a la importancia que tiene la AS para el desarrollo de aplicaciones en la actualidad. David Garlan **(Software architecture: a roadmap, 2000)** establece que la AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño.

Siva Kumar: En la arquitectura de un sistema, se describe la estructura general e ilustra el nivel superior de integración de diseños. También la descripción de la arquitectura incluye el diseño de alto nivel y será como un puente entre las necesidades y la ejecución. **(D. Nielsen, 2000)**

"La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de estos según se la percibe desde el resto de la aplicación y las formas en que interactúan y se coordinan para alcanzar la misión del *software*. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones". **(A Survey of Architecture Description Languages, 1996)**

Una de las definiciones que más se encuentran en la literatura es que la arquitectura es un diseño de alto nivel de abstracción, lo cual hace que se planteé la siguiente interrogante: ¿Qué es abstracción para la arquitectura? Rumbaugh, Shaw y la IEEE **(Reynoso, 2004)** la definen como: la propiedad de identificar los aspectos importantes, o examinar selectivamente ciertos elementos de un problema, posponiendo o ignorando los detalles sustanciales.

Luego de analizar las múltiples definiciones que brindan los especialistas, se maneja para el desarrollo de esta investigación que la Arquitectura de *Software* es la organización fundamental de un sistema, está definida por la estructura comprendida por los componentes de *software* y las relaciones entre ellos. Además de los principios que orientan su diseño, define un estilo o combinación de estilos y patrones para una solución y se considera esencial para el éxito o fracaso de un *software*.

Una vez que se ha conocido el concepto de arquitectura de *software* se hace más fácil entender qué son los estilos arquitectónicos y precisamente, sobre esa y otras temáticas que están muy relacionadas entre sí, estarán dedicados los siguientes epígrafes.

1.3.1 Estilos Arquitectónicos

En algunas bibliografías referentes al estudio de los patrones arquitectónicos se suele llamar de la misma forma a los patrones y estilos, pero la mayoría de los autores los ven de manera separada. **(Reynoso, 2004)**

Los estilos se refieren a formas de estructurar los sistemas y cómo relacionar sus componentes; la diferencia radica en el nivel de abstracción en que se aplican. Los estilos están en un plano de abstracción más alto que los patrones, definiendo cómo configurar la arquitectura, mientras los patrones están más cercanos al diseño.

La selección del estilo arquitectónico depende en gran medida de las necesidades y el alcance del *software*. Un estilo encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los componentes y sus relaciones posibles.

Mary Shaw y Paul Clements definen a los estilos arquitectónicos como un conjunto de reglas de diseño que identifican las clases de componentes y conectores que se pueden utilizar para componer un sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. Los componentes, incluyendo los subsistemas encapsulados, se pueden distinguir por la naturaleza de su computación. **(A field guide to Boxology: Preliminary classification of architectural styles for software systems, 1996)**

Un estilo arquitectónico es una descripción del patrón de los datos y la interacción de control entre los componentes, ligada a una descripción informal de los beneficios e inconvenientes aparejados por el uso del estilo. Los estilos arquitectónicos son artefactos de ingeniería importantes porque definen clases de diseño junto con las propiedades conocidas asociadas a ellos. Estos ofrecen evidencia basada en la experiencia sobre la forma en que se ha utilizado históricamente cada clase, junto con razonamiento cualitativo para explicar por qué cada clase tiene esas propiedades específicas. **(Estilos y Patrones en la Estrategia de Arquitectura de Microsoft, 2004)**

Como pudo apreciarse anteriormente, existen diversos conceptos relacionados con los estilos arquitectónicos, algunos más abarcadores que otros. Estos se agrupan en varias familias de estilos. A continuación se abordan los principales elementos de la más representativa para el desarrollo de la investigación, para más detalles ver ***Anexo 3.***

Estilos de Llamada y Retorno: esta familia enfatiza en los cambios a sistemas o aplicaciones. Son los estilos más generalizados en sistemas a gran escala. Los miembros de esta familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objetos y los sistemas jerárquicos en capas. Se mencionan a continuación las arquitecturas que están dentro de este tipo de estilo debido a su importancia en el desarrollo de sistemas.

- ✓ Arquitectura Modelo Vista Controlador.
- ✓ Arquitectura en Capas.
- ✓ Arquitectura Orientada a Objetos.
- ✓ Arquitectura Basada en Componentes.

A partir del análisis realizado, se decide aplicar dos estilos en la propuesta arquitectónica, pertenecientes a la familia de llamada y retorno. El primero lo constituye el Orientado a Objetos. Entre las ventajas que brinda este estilo se puede mencionar que los objetos están débilmente acoplados, la modificación de ellos puede efectuarse sin comprometer a los otros, pues un objeto es ante todo una entidad reutilizable en el entorno de desarrollo. El segundo estilo seleccionado es la Arquitectura en Capas aplicando el subestilo Cliente/Servidor, debido a que las tareas que se realizarán en el sistema están divididas entre productores y consumidores, además proporciona una amplia reutilización, soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. Como se plantea anteriormente los estilos y los patrones guardan mucha relación, a continuación se abordará más sobre los patrones.

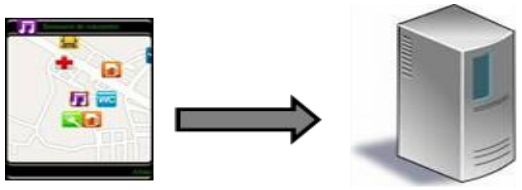

Fig. 1 Estilo Arquitectónico Cliente/Servidor.

1.3.2 Patrones de arquitectura y diseño

En la actualidad existen diversas definiciones de patrones, así como sus clasificaciones. Establecer un patrón para el desarrollo de un *software* posibilita el aprovechamiento de la experiencia acumulada en el diseño de la aplicación. De acuerdo al nivel de abstracción de las vistas de un sistema tienen diferentes clasificaciones donde se pueden mencionar: los patrones de arquitectura y patrones de diseño. El patrón no es más que una solución probada que se puede aplicar con éxito en determinado tipo de problema que se puede repetir en un marco dado.

Como afirmara **(Reynoso, y otros, 2004)**: "Los patrones expresan un esquema de organización estructural para los sistemas de *software*. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y lineamientos para organizar la relación entre ellos"; esto implica que definen la manera en la que el *software* manejará determinada funcionalidad al nivel de la infraestructura.

Patrones de arquitectura

Según **(Reynoso, y otros, 2004)** los patrones arquitectónicos están relacionados con la interacción de objetos dentro o entre niveles, basándose en adaptabilidad a requerimientos cambiantes, rendimiento, modularidad y acoplamiento entre otros problemas arquitectónicos.

El patrón seleccionado es el Orientado a Objetos, este presenta las mismas cualidades que el estilo antes mencionado con el mismo nombre, pero enfocadas al diseño. Este patrón permite que se pueda modificar el sistema de una manera relativamente fácil, pues la modificación de un objeto no implica grandes cambios los restantes objetos. Aunque presenta como desventaja que es necesario conocer la identidad de un objeto para poder invocar sus funcionalidades. Además se aplica el patrón por capas que proporciona una amplia reutilización y permite realizar diferentes implementaciones o versiones de una misma capa.

Patrones de diseño

Proveen un esquema para refinar los subsistemas o componentes de un sistema de *software* o las relaciones entre ellos. Describen la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular. En esta calificación se encuentran los patrones GRAPS (por sus siglas en inglés, *General Responsability*

Asignment Software Patterns) que describen los principios fundamentales de la asignación de responsabilidades a objetos y los GoF (por sus siglas en inglés, *Gans of Four*) que enmarcan los llamados patrones de diseño estructurales, creacionales y de comportamiento. Para conocer más elementos acerca del tema ver ***Anexo 4.***

Patrones de diseño seleccionados

Patrones GRAPS

- ✓ **Experto:** para determinar qué clase debe asumir una responsabilidad a partir de la información que posee. Además permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para realizar los que se le oriente.
- ✓ **Creador:** para guiar la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El diseño bien asignado permitirá soportar una mayor claridad, el encapsulamiento y la reutilización. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.
- ✓ **Controlador:** es un evento generado por actores externos. Se asocian con operaciones del sistema como respuestas a los eventos del sistema, tal como se relacionan los mensajes y los métodos. Un controlador delega en otros objetos el trabajo que se necesita hacer y coordina o controla la actividad. No realiza mucho trabajo por sí mismo.

Patrones GoF

- ✓ **Adapter**: convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases, que de otra manera no podrían tener interfaces incompatibles.
- ✓ **Facade:** sirve para proveer de una interfaz unificada sencilla, que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases. Si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente. Este patrón no oculta las clases sino que ofrece una forma más sencilla de acceder a ellas, en los casos en que se requiere se puede acceder directamente a ellas.

- ✓ **Bridge:** es una técnica usada en programación para desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente sin necesidad de alterar por ello la otra. Esto es, se desacopla una abstracción de su implementación para que puedan variar independientemente.
- ✓ **Observer:** define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- ✓ **Factory Method:** centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
- ✓ **Chain of Responsability:** Delegar responsabilidades a una clase especializada. Permite distribuir la ejecución repartiendo las responsabilidades y constituye la forma básica al realizar un Diseño Orientado a Objetos.

Teniendo conocimiento sobre cuáles son los elementos de la arquitectura que no pueden faltar, es necesario aplicar una metodología que permita generar la documentación necesaria para el futuro desarrollo de la aplicación.

1.4 Metodologías de desarrollo de software

El proceso de creación de un *software* no es una tarea fácil, por lo que existen diferentes metodologías para el desarrollo de los mismos. Para el logro de un producto final de calidad, con el cual estén satisfechos tanto los clientes como el equipo de desarrollo, es necesario la utilización de una metodología, tomando esta como el conjunto de procedimientos, técnicas y el soporte documental que ayuda a los desarrolladores de *software* a obtener un nuevo producto. Actualmente existen muchas metodologías de desarrollo de *software,* tales como RUP (metodología pesada) y XP (metodología ágil). A continuación se analizan sus principales características.

1.4.1 XP (eXtreme Programing)

Es una metodología ágil, centrada en potenciar las relaciones inter personales como clave para el éxito en el desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes y simplicidad en las soluciones implementadas. Esta metodología se define como especialmente

adecuada para proyectos con requisitos imprecisos, muy cambiantes, y donde existe un alto riesgo técnico. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Se puede decir que la metodología XP en su aplicación ideal consta de 6 etapas de trabajo:

- ✓ Exploración
- ✓ Planificación de la Entrega
- ✓ Iteraciones
- ✓ Producción
- ✓ Mantenimiento
- ✓ Muerte del Proyecto

XP es muy recomendada para la programación en pareja debido a que muchos errores son detectados conformes son introducidos en el código (inspecciones de código continuas). Los problemas de implementación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo y varias personas entienden las diferentes partes del sistema. Los programadores conversan mejorando así el flujo de información y la dinámica del equipo, y finalmente, los programadores disfrutan más su trabajo. **(Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP), 2004)**

1.4.2 RUP (Rational Unified Process)

Es una metodología tradicional o pesada. Permite lograr un producto de máxima calidad que cumpla con las necesidades planteadas por el usuario en un tiempo y con un presupuesto acordado con anterioridad. Realiza un modelado visual del *software*, posee una potente documentación y control de cambios.

Sus características permiten que sea adaptable a una gran variedad de sistemas para diferentes áreas de aplicación, distintos tipos de organización y diferentes tamaños de proyecto. La particularidad de que cada ciclo de interacción exige el uso de artefactos, es el motivo que hace que sea una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del *software*.

El ciclo de vida de RUP se caracteriza por ser:

- ✓ Dirigido por casos de uso.
- ✓ Centrado en la arquitectura.

- ✓ Iterativo e Incremental.

RUP divide su ciclo de vida en cuatro fases y nueve flujos de trabajo, de ellos seis de ingeniería y tres de soporte.

Fases de RUP:

- ✓ Inicio: en esta fase se establece la visión y el alcance del proyecto y las partes interesadas deben realizar la estimación de tiempo y costo.
- ✓ Elaboración: en esta etapa se analiza el dominio del problema para establecer una arquitectura base sólida para el desarrollo exitoso del proyecto.
- ✓ Construcción: en esta etapa el objetivo es llegar a obtener la capacidad operacional inicial, así como desarrollar y probar el producto.
- ✓ Transición: el objetivo fundamental es llegar a obtener la liberación del proyecto.

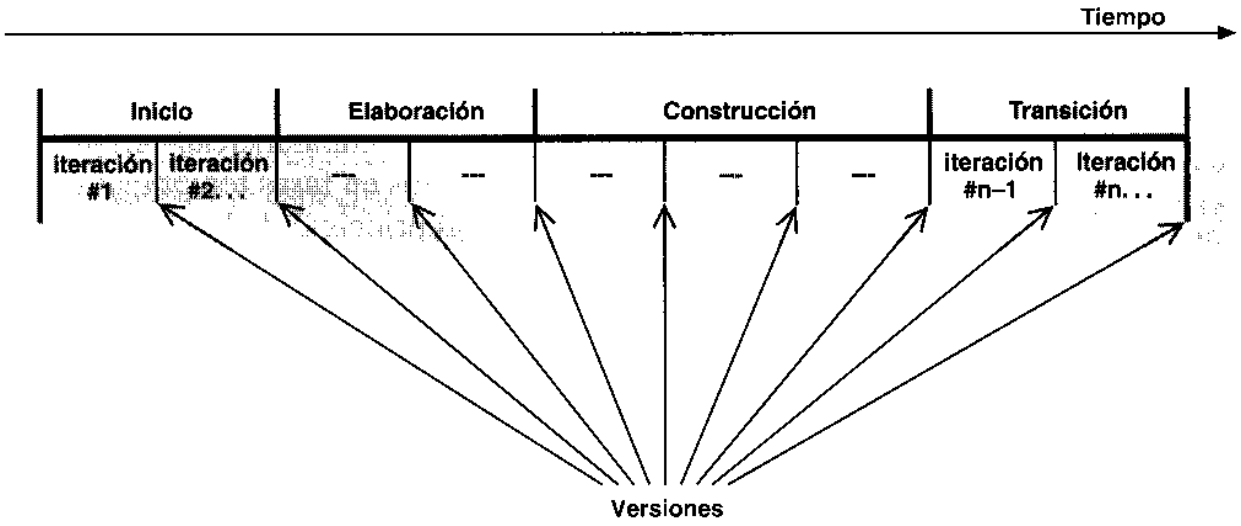

Fig. 2 Fases de RUP. (Jacobson, y otros, 2000)

Flujos de trabajo de ingeniería:

- ✓ Modelamiento de negocio: en este flujo se realiza el entendimiento entre clientes y desarrolladores para concebir las necesidades del negocio que serán abarcadas.
- ✓ Requerimientos: se realiza el acuerdo entre desarrolladores y clientes de lo que el sistema necesariamente debe hacer.

✓ Análisis y Diseño: se realiza una descripción de cómo se implementará el sistema, centrándose en la noción que se tiene de la arquitectura.

✓ Implementación: se crea el *software*, ajustándolo a la arquitectura y asegurando que tenga el comportamiento deseado.

✓ Pruebas: se realizan las pruebas que aseguran que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

✓ Despliegue: el producto final se hace llegar a sus usuarios finales. **(Jacobson, y otros, 2000)**

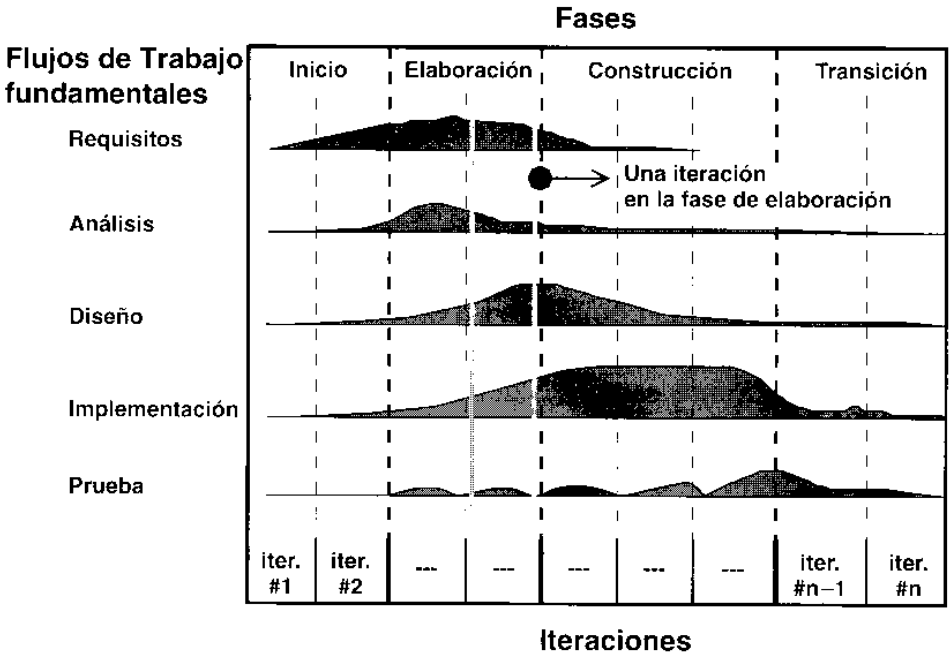

Fig. 3 Flujos de trabajo de RUP. (Jacobson, y otros, 2000)

Luego de haber analizado estas metodologías y teniendo en cuenta que el proceso de desarrollo de *software* es muy complicado y no existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo. Se puede llegar a la conclusión que la metodología más efectiva para el desarrollo del futuro sistema es RUP, considerando que su ciclo de vida está centrado en la arquitectura y además, que permite generar abundante documentación. La misma se elige con el propósito de asegurar la producción de un *software* de alta calidad, que se ajuste a las necesidades de los usuarios finales con un costo y calendario predecibles. Esta metodología establece diferentes roles para el desarrollo de un sistema informático, teniendo como uno de los más importantes el rol de Arquitecto de *Software*. El arquitecto moldea el sistema, diseñándolo de manera que este evolucione.

Para su objetivo el arquitecto trabaja sobre los casos de uso claves en el sistema. Además RUP propone una línea base para el desarrollo.

1.5 Lenguajes de descripción de la arquitectura

El código fuente de un *software* es la forma más exacta de descripción de un sistema, pero no responde a una serie de preguntas que permiten el total entendimiento entre las partes relacionadas en el proceso de desarrollo. Entre ellas se encuentran, por ejemplo, la esencia de los componentes, cuál es su tipo, qué hacen, cómo se comportan, de qué otros componentes dependen y de qué manera, así como qué significan las conexiones y qué mecanismos de comunicación están involucrados, entre otras. Los lenguajes de descripción de la arquitectura (ADL por sus siglas en inglés) surgen ante la necesidad de dar respuesta a estas interrogantes. ***(Anexo 5)***

Los ADL vienen a resolver el problema de la representación formal de la arquitectura y se pueden observar de diferentes maneras. Estos lenguajes pueden ser descriptivos formales o semi-formales, también se pueden encontrar como lenguajes gráficos, incluso en la combinación de los aspectos antes mencionados. **(CAMACHO, et al., 2004)**

Estos se usan de soporte para el análisis y las decisiones tempranas de diseño, proponen que la descripción inicial del sistema puede ser llevada a cabo de forma textual o gráfica. Además, estos lenguajes proveen un mecanismo para la construcción de la arquitectura como artefacto transferible a otros sistemas, de manera tal que pueda ser tomada como marco de referencia o como punto de partida para el resto de las tareas del proceso de desarrollo. **(CAMACHO, et al., 2004)**

En la actualidad existe una gran diversidad de ADL para la representación y prueba de la arquitectura de *software*, sin embargo, el lenguaje seleccionado es UML 2.0, que es el lenguaje propuesto por la metodología RUP. Este es un lenguaje que cuando se revisa la literatura se puede visualizar que en ocasiones se le incluye dentro de los ADL y en otras no. Lo cierto es que este lenguaje cumple con la mayoría de las especificaciones que se tienen en cuenta en el momento de definir los ADL.

UML permite que se puedan representar componentes, así como sus propiedades, interfaces e implementaciones. Tiene la capacidad de representar conectores, así como protocolos, propiedades e implementaciones. Permite la comunicación entre el equipo de desarrollo y los usuarios finales del sistema. Puede ser usado en todas las etapas de desarrollo y es ampliamente conocido en la industria del *software*. Es soportado por diversas herramientas y tecnologías, y también se puede decir que es

ampliamente dominado por el equipo de desarrollo. Presenta un conjunto de herramientas que permiten modelar (analizar y diseñar) sistemas orientados a objetos. Es significativo tener en cuenta que cuanto más complejo es el sistema que se quiere realizar, más importante es el uso de UML.

1.6 Propuesta de Tecnologías y Herramientas a emplear

1.6.1 Herramienta CASE para el desarrollo de software

Las herramientas CASE (Ingeniería de *Software* Asistida por Computadoras) fueron creadas para el desarrollo de la ingeniería de *software* y constituyen un conjunto de aplicaciones destinadas a aumentar la productividad en el avance de sistemas informáticos, reduciendo el coste de los mismos en términos de tiempo y de dinero. Estas se acoplan con las metodologías para representar sistemas y suponen una forma de abstracción del código fuente, a un nivel donde la arquitectura y el diseño se hacen más fáciles de entender y modificar. Cuanto mayor es un proyecto, más necesario es el uso de una tecnología CASE. Entre las herramientas CASE orientadas a UML están: Poseidon for UML, ArgoUML, MagicDraw UML, Borland Together, Enterprise Architec, Rational Rose y Visual Paradigm, siendo las dos últimas las más utilizadas.

Rational Rose

Es comercializada por los desarrolladores de UML y líder en el mundo de modelación visual de sistemas que permite especificar, analizar y diseñar el sistema antes de codificarlo, sin embargo, posee limitantes que la hacen débil en comparación a otras herramientas con las mismas facilidades de modelado, estas debilidades radican en la dependencia de la plataforma Windows y la integración sólo con herramientas que estén en el mismo grupo de *software* propietario. Permite construir los diagramas que se van necesitando durante el proceso de ingeniería según la metodología de desarrollo seleccionada. Es compatible con RUP, brinda muchas facilidades en la generación de la documentación del *software* que se está desarrollando, además posee un gran número de estereotipos predefinidos que facilitan el proceso de modelación. También es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño. Una de las herramientas que supera a **Rational Rose** en estos relevantes puntos es **Visual Paradigm.**

Visual Paradigm 8.0

Está dotado de una buena cantidad de productos o módulos para facilitar el trabajo durante la confección del producto, por lo que se garantiza una buena calidad del producto final. Posee entre sus

principales características la posibilidad de crear un conjunto bastante amplio de artefactos, los cuales se utilizarán durante la confección del sistema. Sus componentes se encuentran relacionados, por lo que se hace muy fácil la creación de cualquier tipo de diagrama, ya que cada componente utilizado en el diagrama que se esté creando, sugiere nuevos posibles componentes a utilizar, y no es necesario localizarlos en la barra donde pueden aparecer un número grande de componentes. Brinda un número considerable de estereotipos a utilizar, lo que permite un mayor entendimiento de los diagramas. Presenta disponibilidad en múltiples plataformas: Microsoft Windows (98, 2000, XP o Vista), Linux, Mac OS X, Solaris o Java. Permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable, y facilita el desarrollo de un proceso cooperativo en el que todos los agentes tienen sus propias vistas de información.

La herramienta seleccionada para realizar la modelación del análisis es el Visual Paradigm 8.0, mediante la cual se puede realizar un diseño centrado en casos de uso y enfocado al negocio que genera un *software* de mayor calidad. Teniendo en cuenta también que la misma presenta el uso del lenguaje estándar UML que fue seleccionado como lenguaje de modelado, lo cual facilita la comunicación del equipo de desarrollo. Visual Paradigm permite realizar una rápida construcción de la aplicación preservando la calidad de la misma y además, presenta características que son favorables para el trabajo con tecnologías libres.

1.6.2 Entorno Integrado de Desarrollo (IDE)

Eclipse

Es una plataforma de código abierto. Generalmente es utilizada para crear entornos integrados de desarrollo. Tiene como lenguaje nativo el Java, además permite el desarrollo en otros lenguajes como son C/C++ y Python. Tiene una arquitectura basada en *plugin* y provee un *framework* para el desarrollo de aplicaciones gráficas*: Graphic Editing Framework (GEF).*

Netbeans

Es una herramienta para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans. Es un producto libre y gratuito sin restricciones de uso. **(Oracle Corporation and/or its affiliates, 2012)**

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de *software* llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (*manifest file*) que lo identifica como módulo. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de *software*.

QT-Creator

Es un entorno de desarrollo multiplataforma ampliamente usado para desarrollar aplicaciones con una interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica tales como herramientas para la línea de comandos y consolas para servidores. Utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de *bindings*. Funciona en todas las principales plataformas Windows y GNU/Linux y tiene un amplio apoyo. El API[6] de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y el manejo de ficheros.

Se propone como entorno de desarrollo utilizar QT Creator, debido a que es el mismo en el que está desarrollada la aplicación Qgis. Entre sus principales características se pueden mencionar que presenta un editor avanzado para C++ y JavaScript, tiene completamiento de código automático y simulador de interfaces para usuarios móviles. Este entorno de desarrollo permite crear aplicaciones de escritorio y plataformas de dispositivos móviles para diversos SO haciendo uso del CMAKE. Está basado completamente en la librería Qt y tiene su código fuente bajo la licencia GNU/GPL, de manera que los proyectos implementados en este entorno deberán ser desarrollados bajo esta licencia.

CMAKE es una herramienta multiplataforma que se utiliza para generar y/o automatizar código. Es decir, constituye un conjunto de herramientas diseñadas para construir, probar y empaquetar software independientemente de la plataforma en la que se desarrolle. Para lograr esto, genera ficheros makefiles nativos y espacios de trabajo que pueden usarse en el entorno de desarrollo deseado. Ello posibilita la generación de ficheros para varios sistemas operativos, lo que flexibiliza el mantenimiento y no se hace necesario poseer varios conjuntos de ficheros para cada plataforma. **(Rodríguez Fuentes, 2011)**

[6] **Interfaz de programación de aplicaciones** o API (del inglés *Application Programming Interface*).

1.6.3 Lenguajes de Programación

C++

Es un lenguaje de programación muy potente, orientado a objetos, del cual se suele decir que es multiparadigma porque permite además la programación estructurada. Tiene la máxima eficiencia al no incorporar verificación de errores en tiempo de ejecución. Es considerado un lenguaje de nivel intermedio, con el cual se puede implementar *software* de bajo nivel, *drivers* y componentes de sistemas operativos, además para el desarrollo rápido de aplicaciones. Este tipo de desarrollo rápido está condicionado según el marco de trabajo, que puede ser el de Borland C++ Builder. Los compiladores de C++ generan código nativo con un alto grado de optimización en memoria y velocidad, lo que lo convierte en uno de los lenguajes más eficientes. Otra fuente de errores es la administración de la memoria, en C++ la asignación y liberación de memoria dinámica es responsabilidad del programador. Se pueden construir mecanismos y jerarquías de clases que controlen correctamente la creación y destrucción de objetos, como así también se pueden escribir programas que no lo hagan, dejando bloques de memoria perdidos.

Python

Es un lenguaje interpretado de alto nivel, muy usado en aplicaciones que necesitan interfaces. Está liberado bajo la licencia *open source* que es compatible con la GNU en su versión 1.2. Es un lenguaje multiparadigma y multiplataforma, ya que se puede utilizar en Windows, Linux/Unix y Mac Os, además que no obliga a los programadores a aplicar un paradigma específico, pues permite la programación orientada a objetos (POO), programación funcional e imperativa. Entre las características de este lenguaje se pueden ver: la Portabilidad, Versatilidad, Interactividad, Sintaxis clara y legible. Es un lenguaje que está en movimiento y pleno desarrollo, pero ya es una realidad y una interesante opción para realizar todo tipo de programas que se ejecuten en cualquier máquina.

Java

Es un lenguaje orientado a objetos desarrollado por la empresa Sun Microsystems. Toma mucho de la sintaxis de C y C++, pero tiene un modelado de objetos más simple y elimina las herramientas de bajo nivel de los lenguajes antes mencionados. Las aplicaciones de java son compiladas en un *bytecode*. Es un lenguaje cuyo código esta liberado bajo la licencia GNU/GPL. Sus aplicaciones pueden ser

ejecutadas sobre cualquier tipo de *hardware*. Es uno de los lenguajes más utilizados para el desarrollo de aplicaciones para los dispositivos móviles.

Los lenguajes seleccionados para el desarrollo de la aplicación son C++ y Python 2.4. Estos son los lenguajes nativos que presenta Quantum GIS. Ambos son multiparadigma. Se puede decir que Python es un lenguaje de alto nivel y de código abierto. Permite escribir nuevos módulos fácilmente en C o C++ y puede incluirse en aplicaciones que necesitan una interfaz programable; mientras que C++, es de nivel intermedio, con el cual se puede implementar *software* de bajo nivel, *drivers* y componentes; genera código nativo con un alto grado de optimización en memoria y velocidad, lo que lo convierte en uno de los lenguajes más eficientes.

1.6.4 Servidor de Mapas

MapServer 5.6.6

Es un motor de procesamiento de datos geográficos escrito en C. Permite crear “mapas de imágenes geográficas”, es decir, mapas que pueden dirigir a los usuarios hacia el contenido. Actualmente MapServer es mantenido por un creciente número de desarrolladores de todo el mundo. Es apoyado por un grupo diverso de organizaciones que patrocinan las mejoras y el mantenimiento, y es administrado al interior de OSGeo.

Salidas cartográficas avanzadas:

- ✓ Ejecución de la aplicación y dibujo de elementos según la escala.
- ✓ Automatización de los elementos del mapa (barra de escala, mapa de referencia y leyenda).
- ✓ Soporte a los lenguajes de scripting y ambientes de desarrollo más populares PHP, Python, Perl, Ruby, Java y .NET.
- ✓ Soporte multiplataforma Linux, Windows, Mac OS X, Solaris.
- ✓ Soporte a un gran número de estándares del OGC: WMS (cliente/servidor), WFS no-transaccional (cliente/servidor), WMC, WCS, SLD, GML, SOS, OM.
- ✓ Múltiples formatos de datos vectoriales y ráster (TIFF/GeoTIFF, EPPL7 y otros por medio de GDAL).
- ✓ Archivos shapefile de ESRI, PostGIS, ESRI ARCSDE, Oracle Spartial, MySQL.
- ✓ Soporte de proyecciones cartográficas.

- ✓ Proyección de mapas “al vuelo” con miles de proyecciones disponibles a través de la librería Proj. 4 **(Regents of the University of Minnesota)**

GeoServer

Es un servidor de código abierto escrito en Java que permite a los usuarios compartir y editar datos geoespaciales. Diseñado para la interoperabilidad, publica datos de cualquier gran fuente de datos del espacio usando estándares abiertos. Sirve de implementación de referencia del estándar OGC *Web Feature Service*, y también implementa las especificaciones de *Web Map Service* y *Web Coverage Service*.

Entre las principales características de GeoServer se pueden citar algunas como:

- ✓ Enteramente compatible con las especificaciones WMS, WCS e WFS.
- ✓ Fácil utilización a través de la herramienta de administración vía web no es necesario entrar en archivos de configuración grandes y complicados.
- ✓ Soporte amplio de formatos de entrada PostGIS, Shapefile, ArcSDE y Oracle.
- ✓ Soporte de formatos de salida tales como JPEG, GIF, PNG, SVG y GML.
- ✓ Soporte para edición de datos de banco de datos individuales a través del protocolo WFS *transactional profile* (WFS-T), disponible para todos los formatos de datos. Basado en *servlets* Java (JEE), puede funcionar en cualquier *servlet* contenedor. Proyectado para ser compatible con extensiones **(OsGeo).**

MapServer 5.6.6

Es el servidor de mapas seleccionado. Es un motor de procesamiento de datos geográficos escrito en lenguaje C. Es multiplataforma, que brinda soporte a un gran número de estándares del OGC como se pueden mencionar entre otros el Web Map Server y el *Geography Markup Language* (GML, por sus siglas en inglés), destinado al modelaje, transporte y almacenamiento de la información geográfica. Puede soportar tipos de datos tanto vectoriales como ráster y también shapefile. Además de soportar numerosos Sistemas Gestores de Base de Datos (SGBD)**.**

1.7 Conclusiones

Después de haber estudiado y profundizado en el desarrollo de SIG para dispositivos móviles y teniendo en cuenta las tecnologías y herramientas más usadas en la confección de estos sistemas, se

puede concluir que el desarrollo de este tipo de sistemas es muy heterogéneo, pues no presentan puntos en común para el proceso de construcción de los mismos.

La selección de Quantum GIS como base del desarrollo permitirá reducir el tiempo de implementación, pues este cuenta con las librerías que brindará el soporte lógico. Se ha escogido como lenguaje de modelado a UML, que a pesar de no ser un ADL sí cumple con las condiciones fundamentales para modelar los artefactos que se obtendrán como resultado de esta investigación. Como metodología de desarrollo se seleccionó RUP y como herramienta para el modelado Visual Paradigm. Además como IDE se propone utilizar QT Creator, el cual admite los lenguajes de programación C++ y Python que son los lenguajes escogidos y forman parte la base de Qgis.

Del conocimiento adquirido durante la investigación se decide que la propuesta debe hacerse funcional en cualquier dispositivo móvil que tenga como sistema operativo *Symbian* o *Andriod*, pero sobre todo en los teléfonos inteligentes, herramientas que cada vez son de mayor uso popular. Después del estudio realizado donde se identificaron los elementos necesarios para el diseño de la arquitectura, se puede dar paso a la descripción de la misma en el próximo capítulo.

Capítulo II

Descripción de la arquitectura

La Arquitectura de *Software* es el diseño de más alto nivel de la estructura de un sistema. Se selecciona y diseña con base a los requisitos y a los atributos de calidad. Los requisitos son aquellos prefijados para el sistema de información geográfica, pero no solamente los de tipo funcional, sino también los no funcionales como la modificabilidad, seguridad, flexibilidad e interacción con otros sistemas de información. Los componentes llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos.

En el presente capítulo se muestran los requisitos que son críticos para la arquitectura. Se explica el flujo de procesos además de exponerse las vistas arquitectónicas (4+1 vistas) representadas con el lenguaje de modelado UML. A partir de las temáticas abordadas en el capítulo anterior y la selección de las herramientas a utilizar, es posible realizar la propuesta de la arquitectura que proporcionará solución al problema propuesto en el diseño teórico planteado para esta investigación.

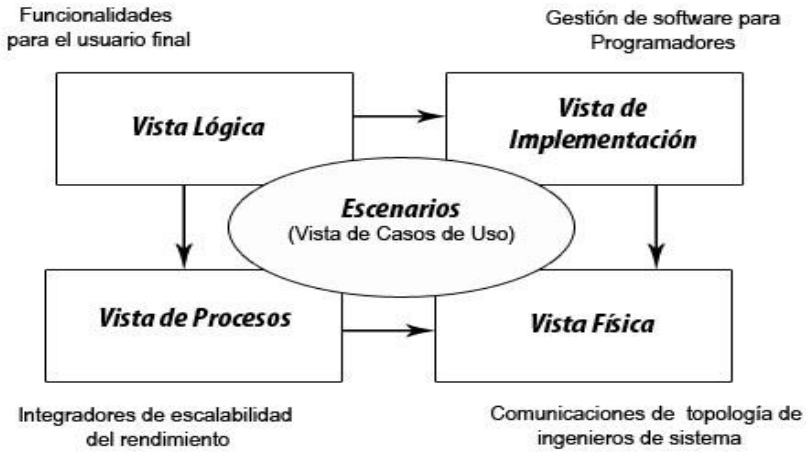

Fig. 4 Diagrama 4+1 Vistas.

2.1 Atributos de calidad y requisitos que tienen impacto sobre la arquitectura

A lo largo del proceso de diseño y desarrollo de una aplicación, los atributos de calidad juegan un papel importante ya que estos generan las decisiones sobre el diseño y los argumentos que los justifican. Según la literatura consultada un *software* de calidad debe satisfacer los requisitos dados por el usuario. Las características adicionales o atributos de calidad de un sistema están estrechamente

relacionadas con el uso que se le dará en la arquitectura que se obtendrá como resultado de esta investigación.

Calidad, según Pressman es “la concordancia con los requisitos funcionales y de rendimiento establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo *software* desarrollado de forma profesional”. **(Arquitectura de Software, 2004)**

Los atributos de calidad se pueden organizar y descomponer de maneras diferentes, en lo que se conoce como modelos de calidad. Existen diferentes modelos de calidad, cada uno con sus características propias. De los modelos consultados, el ISO/IEC 9126 es el más acorde a la arquitectura que se propone para efectos de la evaluación de arquitecturas de *software*. El modelo se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad.

Característica	**Subcaracterística**	**Elementos de tipo arquitectónico**
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia.
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos.
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos.
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea.
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de *software* para manejar excepciones.
	Recuperabilidad	Existencia de mecanismos o dispositivos de *software* para restablecer el nivel de desempeño y recuperar datos.
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad.
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo.
Mantenibilidad	Acoplamiento	Interacciones entre componentes.
	Modularidad	Número de módulos que dependen de un

		componente.
Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación.
	Instalabilidad	Presencia de mecanismos de instalación.
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia.
	Reemplazabilidad	Lista de elementos reemplazables para cada componente.

Tabla 3. Atributos de calidad adaptados para arquitecturas de software.

Atendiendo a los atributos de calidad antes relacionados, se han identificado los que se van a tener en cuenta para la propuesta de arquitectura:

Confiabilidad: se previenen los fallos de *software* realizando la especificación de requisitos, los métodos de diseño comprobados, utilizando lenguajes con abstracción de datos, utilización de herramientas CASE adecuadas para gestionar los componentes. La separación de los servidores del cliente proporciona también un alto grado de confiabilidad puesto que las fallas en uno no repercuten directamente en el otro. Se realizarán pruebas que garantizarán la veracidad y robustez del código. Con la correcta aplicación de este atributo se satisfacen principalmente tres necesidades: que el sistema cumpla con las especificaciones, es decir, que sea consistente con éstas; que tenga la capacidad de reaccionar bien ante situaciones excepcionales (tolerancia a fallas) y recuperarse en caso de fallas en el menor tiempo posible; y la capacidad de estar operativo el mayor tiempo posible. Todo proporciona al sistema una mayor confiabilidad.

Eficiencia: en este sentido a la aplicación le corresponde permitir que la información almacenada pueda ser consultada, sin que se afecte el tiempo de respuesta. El sistema debe poseer la capacidad de dar respuesta a las peticiones de cada usuario, en un tiempo máximo de 3 segundos. Lo importante es que la aplicación sea implementada manteniendo un balance adecuado entre eficiencia y corrección. Obteniendo una buena eficiencia se logra que el *software* tenga la capacidad de hacer buen uso de los recursos que manipula.

Mantenibilidad: es la capacidad del producto de *software* para ser modificado. Las modificaciones pueden incluir el aumento de funcionalidades, mejoras o adaptaciones del *software* a los cambios del entorno y de los requisitos funcionales. En la propuesta de arquitectura que se realiza se garantiza el cumplimiento de este atributo de calidad, lo cual se demuestra por ejemplo con la utilización de la

arquitectura orientada a objetos, la cual permite que los objetos estén débilmente acoplados y la modificación de ellos puede efectuarse sin comprometer a los otros y de igual forma con la arquitectura en capas, con su subestilo cliente/servidor, que permite cierta independencia entre las capas, dando la posibilidad de realizar cambios en una sin afectar a la otra.

Portabilidad: el sistema podrá ser usado en varias plataformas móviles y sobre un entorno de desarrollo que posibilite generar aplicaciones que puedan ser instaladas y ejecutadas en los diferentes sistemas operativos de estos dispositivos. La arquitectura que se propone permite que la herramienta a desarrollar pueda ser ejecutada en cualquier *smartphone* que tenga los sistemas operativos Symbian o Android. Se hace posible que la facilidad de instalación que permite reemplazar una nueva versión del sistema por la existente, sin impacto en el mismo o en sus componentes. Estos elementos posibilitan la instalabilidad, reemplazabilidad, adaptabilidad y coexistencia del sistema.

Además de los atributos de calidad definidos en el modelo anteriormente expuesto, se tienen otros elementos que son de vital importancia para proponer la arquitectura adecuada en esta investigación, como son los requisitos no funcionales que se relacionan a continuación.

2.2 Identificación de los requisitos de software

En lo que a requisitos funcionales y no funcionales se refiere, las opiniones son bien diversas, en cuanto a cuáles son los que rigen el proceso de diseño de la arquitectura. Lo cierto es que desde la visión abstracta de la arquitectura: la seguridad, escalabilidad o la confiabilidad tienen una gran importancia, así como también la tienen los requisitos funcionales que responden a las necesidades que puedan tener los clientes y los cuales permiten lograr la organización fundamental de la aplicación según la metodología escogida.

2.2.1 Requisitos no funcionales del sistema

Los requisitos no funcionales son propiedades o cualidades que la aplicación debe tener. Es decir, son las características que hacen al producto atractivo, usable, rápido o confiable. La aplicación que se propone en esta investigación debe cumplir los siguientes requisitos no funcionales.

Requisitos de Usabilidad

- ✓ La aplicación puede ser utilizada por usuarios con conocimientos básicos en tecnología móvil.

✓ El *software* tendrá siempre visible la opción de Ayuda, lo que posibilitará un mejor aprovechamiento de sus funcionalidades por parte de los usuarios.

Requisitos de Fiabilidad

✓ El sistema debe estar disponible en todo momento para sus usuarios, descontando el tiempo en que se encuentre en mantenimiento.

Requisitos de Soporte

✓ El mantenimiento que recibirá la aplicación se efectuará en el período de tiempo determinado por el equipo de desarrollo y los clientes.

Requisitos de Interfaz

✓ Debe contar con un diseño gráfico orientado a las características de los dispositivos móviles y sin uso de múltiples imágenes.
✓ El tamaño de la ventana debe ser apropiado para las limitaciones de memoria y capacidad de visualización de los dispositivos móviles.
✓ El sistema debe mostrar de forma organizada sus funcionalidades.

Requisitos de Software

✓ Cliente

a. Sistema Operativo Android o Symbian.

✓ Servidor de mapas

a. Sistema Operativo Windows XP o GNU/Linux Ubuntu.
b. MapServer 5.6.6 o superior.

Requisitos de hardware

✓ Cliente

a. Pantalla *Touch* y/o teclado de tipo Qwerty[7].
b. Procesador 1.0 GHz como mínimo.
c. *Chipset*.
d. Memoria interna de 4 GB como mínimo.
e. Tecla para la navegación.

- ✓ Servidor de mapas

a. Se requiere al menos 1GB de memoria RAM.
b. Se requiere al menos 40GB de disco duro.
c. Procesador 2.2 GHz o superior.
d. Procesador Pentium IV o superior.
e. Tarjeta de red (tipo de conexión: inalámbrica).

2.2.2 Requisitos funcionales

Los requisitos funcionales especifican comportamientos particulares de un sistema. Pueden ser cálculos, detalles técnicos, manipulación de datos o funcionalidades específicas. Los requisitos que fueron identificados para esta herramienta son los siguientes:

Control de Capas

✓ El sistema debe permitirle al usuario poder agregarle capas al mapa base.
✓ El sistema debe permitirle al usuario seleccionar las capas del mapa que desee visualizar en el dispositivo.
✓ El sistema debe permitirle al usuario desmarcar las capas que no desee visualizar.
✓ El sistema debe permitirle al usuario agregar capas ráster y vectoriales.
✓ El sistema debe permitirle al usuario agregar capas WMS y WFS.

Navegación

✓ El sistema debe permitirle al usuario realizar diferentes acciones en el mapa, transformando el nivel de referencia con que se visualiza.

[7] **Qwerty:** permite la escritura de manera más cómoda y fácil, como si se escribiera con el teclado convencional de la computadora.

- ✓ El sistema debe permitirle al usuario desplazar el mapa a la izquierda o a la derecha de la pantalla sin modificar la escala del mapa.
- ✓ El sistema de permitirle al usuario desplazar el mapa hacia la parte superior o inferior de la pantalla sin modificar la escala del mapa.
- ✓ El sistema de permitirle a usuario aumentar el mapa y transformar el nivel de referencia.
- ✓ El sistema de permitirle a usuario disminuir el mapa y transformar el nivel de referencia.

Análisis

- ✓ El sistema debe permitirle al usuario realizar diferentes selecciones sobre el mapa.
  - ➢ El sistema debe permitirle al usuario realizar selecciones de áreas sobre el mapa sin modificar la escala.
  - ➢ El sistema debe permitirle al usuario la selección por puntos sobre el mapa sin modificar la escala.
- ✓ El sistema debe permitirle al usuario realizar cálculo de distancia entre dos o más puntos sobre el mapa.
- ✓ El sistema debe permitirle al usuario realizar localizaciones sobre el mapa.
  - ➢ El sistema debe permitirle al usuario realizar localizaciones de objetos georefenciados.
  - ➢ El sistema debe permitirle al usuario realizar localizaciones en un área determinada.
- ✓ Es sistema debe permitirle al usuario identificar un objeto en el mapa.
- ✓ El sistema debe brindarle al usuario la posibilidad de editar las capas que desee del mapa.
  - ➢ El sistema debe permitirle al usuario agregar un punto, una recta o un polígono al mapa.

Archivo

- ✓ El sistema debe permitir que se le agreguen nuevos complementos.
- ✓ El sistema debe permitir que se puedan exportar los mapas.

Una vez determinadas las funcionalidades que presentará la aplicación. Se puede comenzar el diseño de los prototipos de interfaz donde se representarán estas funcionalidades.

2.3 Arquitectura de la Información

De los principales problemas que se presentan en el momento de construir una aplicación para dispositivos móviles y en especial para los teléfonos, es de cuánto será el tamaño de la ventana o cómo se verá la aplicación. Aunque en la actualidad estos dispositivos son cada vez más parecidos a las computadoras en cuanto a su capacidad de procesamiento y tamaño en memoria. Las respuestas a estas preguntas se encuentran en el desarrollo de una Arquitectura de Información (AI). Este es un término que se comienza a utilizar en la década del 70 y es la encargada de diseñar, organizar, distribuir la información no solo en un sitio web sino en otras esferas que se necesita hacer un buen uso de la información. Para el desarrollo de la AI, se pueden utilizar diferentes técnicas con el fin organizar el contenido que se le va a ofrecer a los usuarios. En el caso de los prototipos de interfaz que se proponen, se realizan utilizando la técnica que se muestra a continuación.

2.3.1Técnica de interacción con el contexto

Esta técnica consiste en realizar un estudio acerca de los productos similares al que se desea desarrollar y si es posible, mejorar los que ya existen, además que se pueda fijar en las deficiencias de los demás y en base a ello trabajar para no cometer los mismos errores. Se realizó un estudio de las interfaces presentadas por ArcGis Mobile, GvSIG Mini, GvSIG Mobile y otros Sistemas de Información Geográfica para dispositivos móviles, con el fin de evaluar la estructura de su diseño. Estas herramientas en su diseño de interfaz son muy parecidas, pues todas presentan las mismas características solo variando en los colores y en donde ubican los contenidos.

GvSIG Mobile

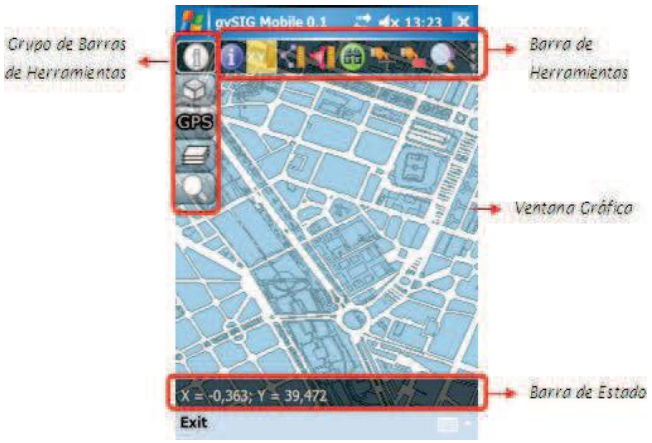

Fig. 5 Interfaz de GvSIG Mobile (Montesinos Lajara & Carrasco Marimon).

Como se muestra en la fig. 5, este presenta una barra de título en la parte superior de la pantalla, donde se refleja el nombre de la aplicación y la versión. Además tiene una barra de herramientas justo debajo y presenta un menú lateral con las opciones de las diferentes barras de herramientas que se pueden seleccionar. Entonces muestra el área de trabajo, donde se encontrará el mapa.

ArcGis Mobile

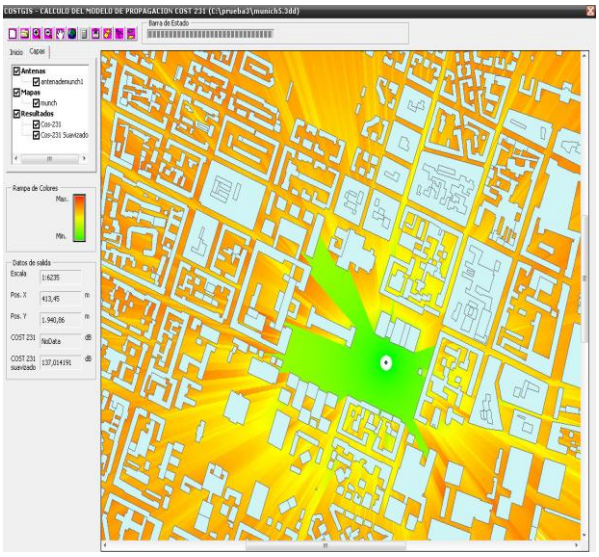

Fig. 6 Interfaz de ArcGis Mobile (Díaz Salvador, 2010).

En la fig. 6 se presenta una barra de título donde se muestra el título de la aplicación y la opción de salir del sistema. En la barra de menú principal se encuentran las principales funcionalidades que brinda el *software* a los usuarios. En el menú lateral izquierdo se publica el árbol de las capas que son seleccionables, el cuadro de la rampa de colores y el cuadro de estado.

Este patrón de interfaz que se observa en estos SIG los presentan además QvSIG Mini, MapWindow y ArcPad, por lo que se decide realizar una propuesta de interfaz de usuario que reúna algunas de estas características e incorpore otras propias de la aplicación.

El primer elemento a tener en cuenta es que se desea desarrollar una aplicación que consistiría en un Visor de Servicios de Mapas de Quantum GIS (VMS QGIS), una aplicación de escritorio nativa dirigida a los *Smartphone*. Como estos dispositivos se encuentran en el tercer grupo de dispositivos móviles y

según las funcionalidades que estos ofrecen, queda definido que las ventanas de la aplicación que se muestre a los usuarios serán pantallas de tamaño mediano a grandes (por encima de los 240 x 120 píxeles). Además, presentarán un diseño sencillo, sin tantas imágenes que hagan a la aplicación consumir muchos recursos de memoria del móvil. La información debe encontrarse distribuida de tal manera que con el máximo de tres selecciones, el usuario llegue a la funcionalidad deseada de las que brinda la herramienta. Las barras que presentará la aplicación, menos la del título, se ocultarán automáticamente cuando no estén activadas por el usuario así quedará más espacio para la visualización del mapa. Los prototipos que se realizan se basan en la propuesta de la vista de presentación del expediente de arquitectura de la UCI, con el modelo de CMMI[8] nivel 2.

2.3.2 Representación visual de contenidos de la pantalla de Bienvenida

Elementos que deben estar ubicados en la pantalla de bienvenida.

- ✓ Logotipo.
- ✓ Nombre de la aplicación: Visor de Servicios de Mapas de QGIS para Dispositivos Móviles.

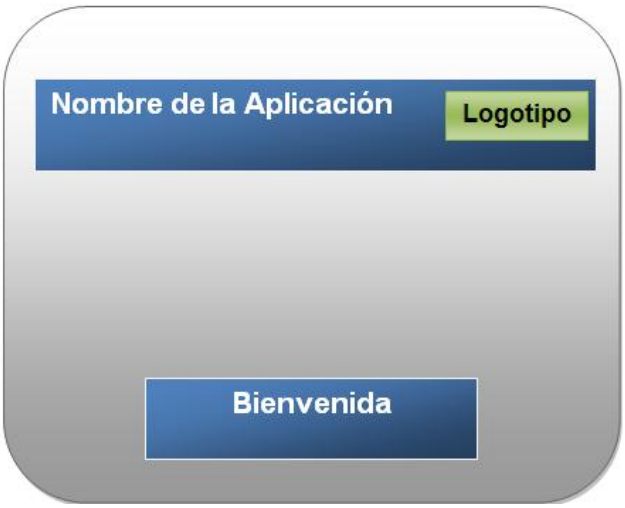

Fig. 7 Pantalla de Bienvenida.

Al acceder al sistema se debe mostrar la pantalla de Bienvenida. Esta pantalla se conforma del logotipo, el nombre del sistema y el mensaje de bienvenida.

[8] **CMMI:** Modelo de Madurez de Capacidad Integrado, pertenece a la familia de modelos desarrollados por el SEI (Software Engineering Institute) para evaluar las capacidades de las organizaciones de ingeniería de sistemas, ingeniería de software, además del desarrollo integrado del producto y del proceso.

2.3.3 Distribución de áreas en pantalla genérica

Barra de Título: contiene los recursos informativos del sistema como son el logotipo y el nombre del sistema.

Barra de Menú Principal: menú que incorpora los contenidos específicos del sistema. Los contenidos deben estar organizados de manera que no se sobrecargue el menú para facilitar el acceso desde el punto de vista de usabilidad. En caso de que los contenidos sean numerosos deberán ser agrupados bajo algún criterio de clasificación que permita mostrarlos sin necesidad de *scroll*.

Área de Trabajo: área donde tiene lugar el desarrollo de las tareas o funcionalidades inherentes al sistema.

Forma de Desplegar el Menú: desplegable al seleccionar la opción que se desee. Solo puede tenerse una opción del menú desplegada de manera que cuando se seleccione otra opción desplegable fuera de esa jerarquía, la anterior debe replegarse.

Menú Lateral Izquierdo: mostrará un árbol donde se encontrarán las capas que podrá habilitar o no el usuario.

Fig. 8 Distribución de áreas en pantalla.

2.3.4 Distribución del contenido en la pantalla genérica

En la barra de título se muestra el logotipo y el nombre del sistema. En la barra de menú principal se encontrarán las funcionalidades que tendrá la aplicación agrupadas por módulos según la función que realicen. En el menú lateral izquierdo se mostrará un árbol con todas las capas que puede seleccionar el usuario para visualizarlas. En el área de trabajo se mostrará el mapa en cual se encontrarán representados los objetos georeferenciados.

Fig. 9 Distribución de contenido en pantalla.

En la barra de menú principal cada opción contendrá un grupo de submenú. En todos los enunciados de los submenús se encontrará un número para facilitar el acceso rápido a las funcionalidades. Esta distribución se muestra a continuación en la Fig. 10, donde se mostrarán los submenús del módulo de navegación.

Fig. 10 Submenú del módulo de navegación.

A continuación se hará una breve descripción de cada uno de los submenús, para un mejor entendimiento de su significado.

Paneo_N: con esta opción el usuario podrá mover el mapa hacia arriba, abajo, a la derecha y a la izquierda simplemente seleccionando el mapa y arrastrándolo o presionando la tecla de navegación del dispositivo.

Aumentar_A: el usuario podrá aumentar el tamaño del mapa en la región seleccionada y tendrá la posibilidad de acceder a esta opción presionando la tecla A del teclado.

Disminuir_D: el usuario podrá disminuir el tamaño del mapa en la región seleccionada y tendrá la posibilidad de acceder a esta opción presionando la tecla D del teclado.

De esta manera quedará distribuido el contenido que ofrecerá la aplicación a los usuarios. Cada una de estas ventanas podrá ser personalizada en dependencia del negocio.

Luego de haber definido los prototipos de las interfaces principales, se prosigue el desarrollo de la investigación con el estudio de las vistas arquitectónicas propuestas por Kruchten.

2.4 Vistas de la Arquitectura

En las vistas de la arquitectura se recogen los elementos fundamentales de la propuesta que se realiza. A continuación se muestra una descripción de cada una de las vistas donde se pone de manifiesto el alcance de cada una de ellas.

2.4.1 Vista de casos de usos

Esta vista está formada por los casos de uso que son críticos para el desarrollo de la aplicación. En ella se representan las necesidades funcionales que presenta la herramienta. Es considerada la base para las demás vistas que complementan el marco escogido, que son la vista lógica, la vista de implementación, la vista de procesos y la vista de despliegue. Con los casos de uso que son representados en esta vista se podrá validar la arquitectura propuesta.

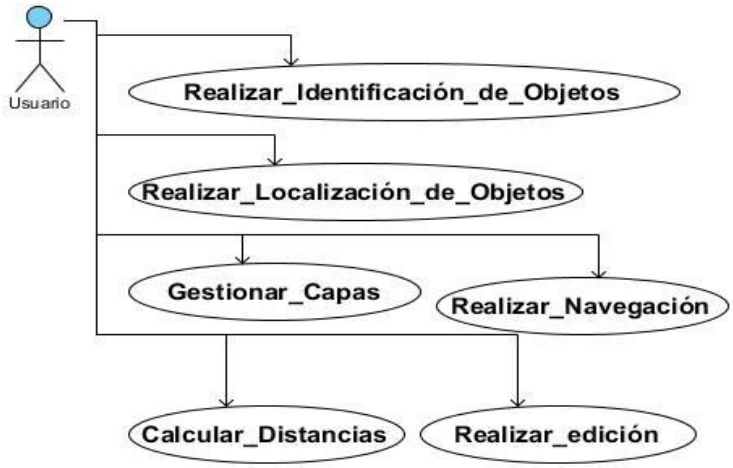

Fig. 11: Diagrama de casos de uso críticos para el sistema.

Para un mejor entendimiento del diagrama se dará una breve descripción de cada caso de uso representado.

Gestionar Capas: este caso de uso le brinda la posibilidad al usuario de seleccionar y agregar al mapa las capas que desee visualizar. Estas capas pueden ser de tipo vectoriales, ráster, WMS y WFS.

Realizar Navegación: con este caso de uso se podrán realizar diferentes acciones en el mapa, que permitirán obtener mejores y más detalladas vistas de un objeto como son: Zoom in (Aumentar), Zoom out (disminuir) y Paneo (mover el mapa).

Realizar Localización de Objetos: este caso de uso representa la funcionalidad que brinda el sistema de realizar búsqueda de un objeto referenciado y localizarlo posteriormente en el mapa.

Realizar Identificación de Objetos: este caso de uso permitirá conocer los datos asociados a un objeto georeferenciado.

Calcular Distancia: este caso de uso brindará la posibilidad de conocer la distancia entre dos o más puntos en el mapa.

Realizar Edición: este caso de uso le brinda la posibilidad de hacer diferentes ediciones sobre el mapa, ya sea agregar un nuevo punto o crear una capa nueva.

2.4.2 Vista Lógica

En el enfoque arquitectónico 4+1 vistas, la vista lógica es la que muestra la estructura del sistema. Mediante esta se muestra a los usuarios las funcionalidades que contiene la aplicación. En ella quedan identificados los mecanismos y se diseñan los elementos comunes a través de la aplicación, principalmente los requisitos funcionales del sistema, funciones y servicios que se han definido. Se utiliza el estilo arquitectónico orientado a objetos y la notación empleada es UML.

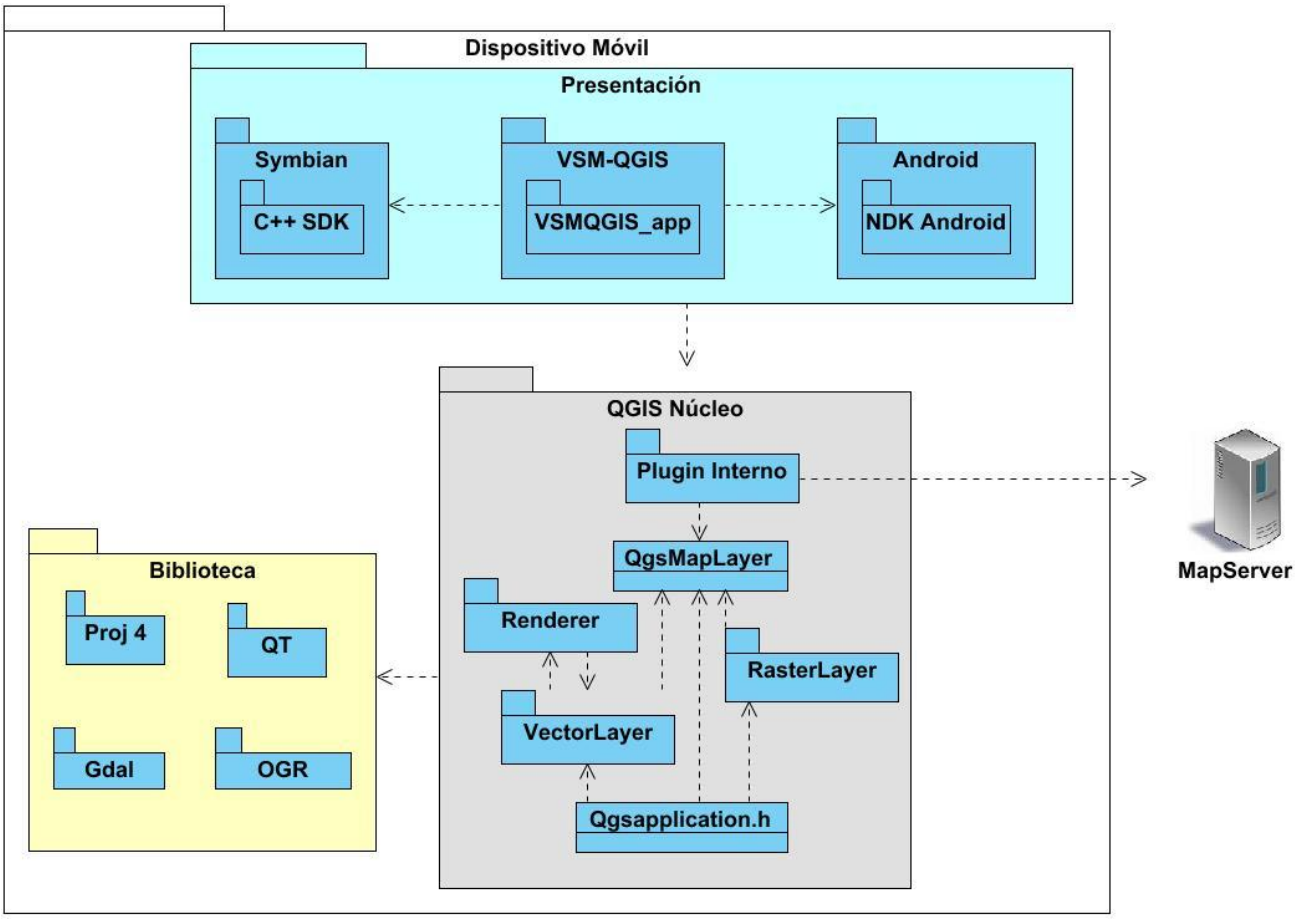

Fig. 12 Vista Lógica.

De forma general se muestra en la Fig. 12 el funcionamiento lógico del Visor de servicios de mapas de Quantum GIS para dispositivos móviles (VSM QGIS). Esta será una aplicación que tendrá todos los requisitos necesarios para ser nativa de los Smartphone que tengan los sistemas operativos Android o Symbian. Para el desarrollo de este visor se utilizan varias bibliotecas como son: las librerías que brindan el API de Android y el Kit de Desarrollo de Software (SDK, por sus siglas en inglés) de Symbian. Otra biblioteca que se emplea es el núcleo de Quantum GIS, con la cual se construyen las

capas que se mostrarán, además de encargarse de la comunicación con el servidor. Que esta aplicación pueda ser nativa de estas plataformas no es casual, pues así se mejora el rendimiento de estos dispositivos.

2.4.3 Vista de Implementación

Esta vista comprende a grandes rasgos todos aquellos artefactos que se utilizan para ensamblar el sistema y ponerlo en producción, ya listo para su distribución física. Muestra la organización y las dependencias entre un conjunto de componentes. **(Anexo 6)**

El software se empaqueta en partes pequeñas, bibliotecas de programas o subsistemas que pueden ser desarrollados por uno o un grupo pequeño de desarrolladores. Los subsistemas se organizan en una jerarquía de capas, cada una de las cuales brinda una interfaz estrecha y bien definida hacia las capas superiores. **(Kruchten, 2006)**

En la vista de implementación (ver Fig. 13) se muestran los principales componentes del sistema. Estos se encuentran agrupados por paquetes con el fin de lograr una organización lógica.

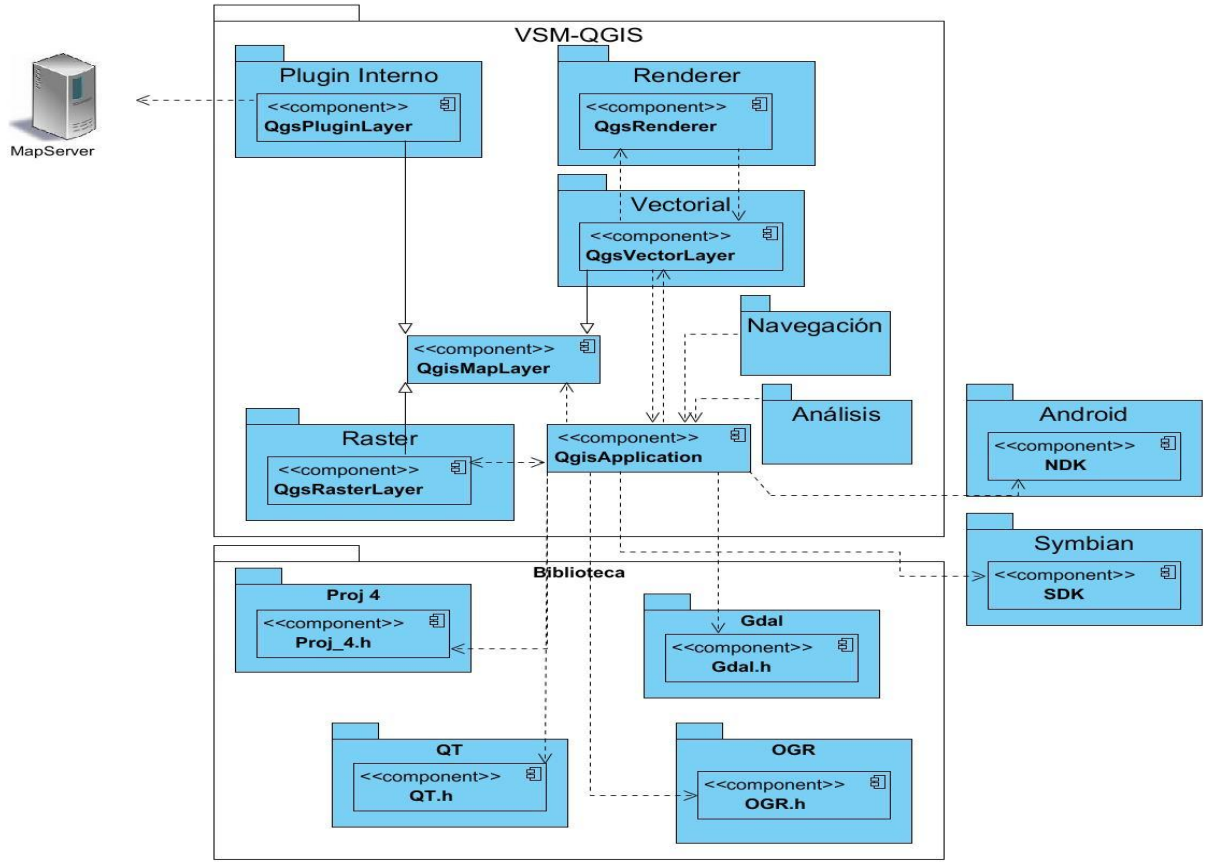

Fig. 13 Vista de Implementación.

El paquete Android brinda un conjunto de componentes que apoyarán el desarrollo de la herramienta con mayor calidad y eficiencia. (Ver Fig. 14)

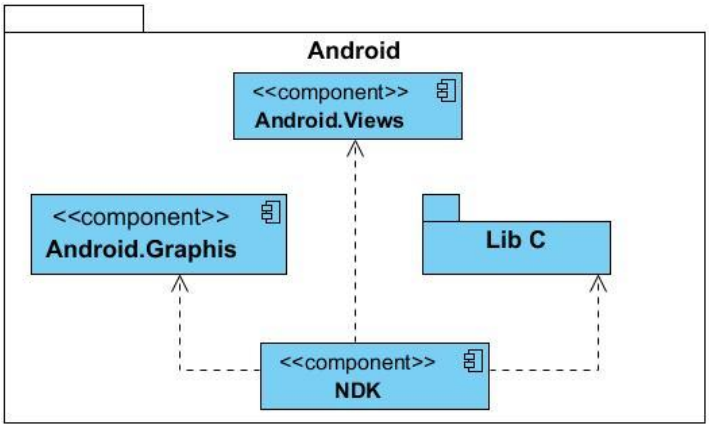

Fig. 14 Paquete Android.

Los componentes representados en la Fig. 14 serán explicados brevemente según la funcionalidad que realizan.

NDK: kit de desarrollo nativo, es el kit que utiliza Android para el desarrollo de aplicaciones con el lenguaje C++.

Android.Views: este componente permite la interacción con el usuario.

Android.Graphis: es el encargado de los gráficos de la API de suministro de las clases de bajo nivel de gráficos, los colores y además permite dibujar.

Lib C: esta es una biblioteca de C estándar optimizada para Linux, basada en dispositivos embebidos.

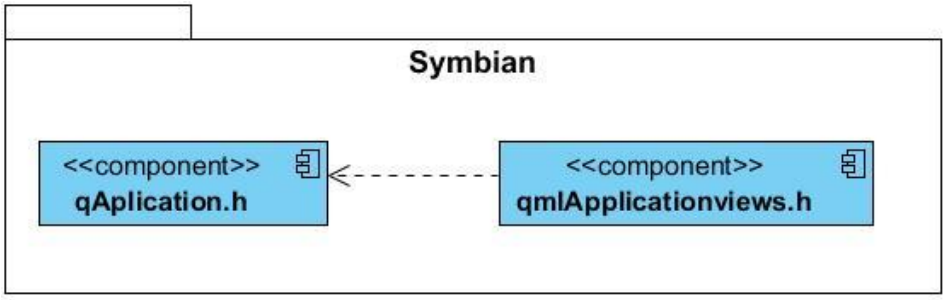

Fig. 15 Paquete de Symbian.

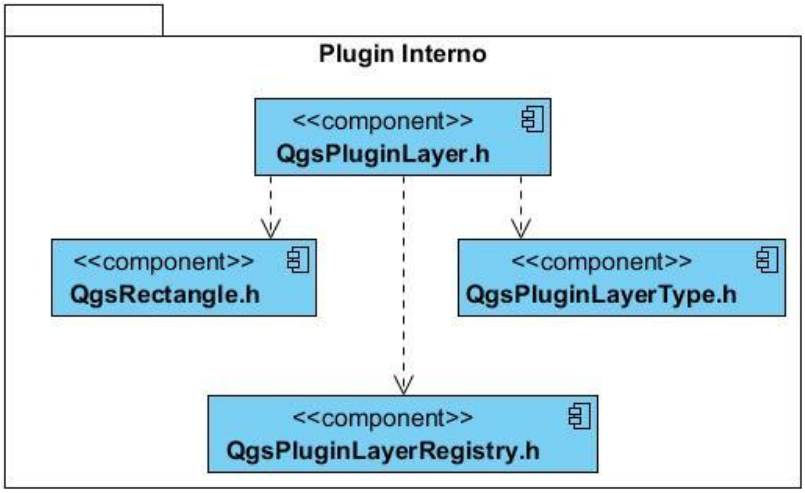

Fig.16 Paquete de Plugin Interno.

Plugin Interno: este paquete posibilita la integración de *plugins* al sistema para ampliar sus funcionalidades. Además, como propuesta se plantea que en este paquete se cree la estructura para realizar a través de él los pedidos al servidor de mapas.

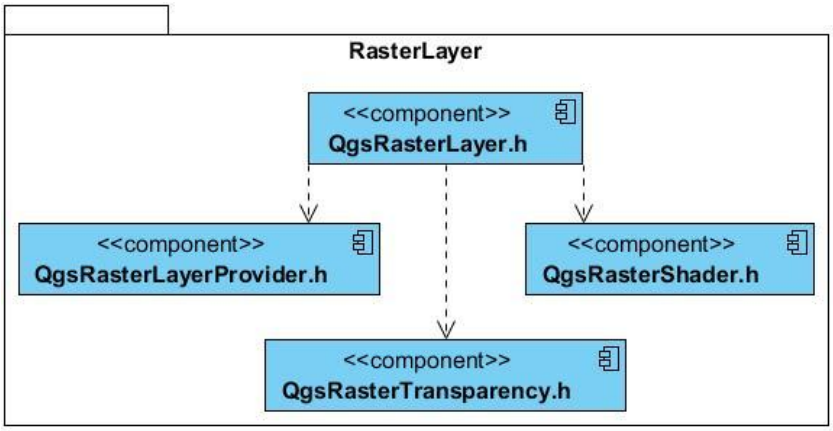

Fig. 17 Paquete de Raster Layer.

RasterLayer: en este paquete se encuentran las clases que se encargan de las capas ráster, las cuales proporcionan a la aplicación la capacidad de representar conjuntos de datos ráster y contener la información asociada. La clase QgsRasterLayer, su clase principal, hace uso de GDAL y apoya por lo tanto cualquier formato GDAL compatible.

2.4.4 Vista de Procesos

Muestra los hilos y procesos de ejecución así como la comunicación entre estos. La vista de procesos toma en cuenta algunos requisitos no funcionales tales como el rendimiento y la disponibilidad. Se enfoca en asuntos de concurrencia y distribución, integridad del sistema y de tolerancia a fallas.

Esta vista se realiza cuando la aplicación debe soportar la concurrencia de varios hilos de ejecución. Por lo que no es necesario representarla en esta aplicación debido a que esta herramienta se encontrará disponible en cada dispositivo móvil y solo la utilizará un usuario a la vez.

2.4.5 Vista de Despliegue

En la vista de despliegue se toman en cuenta primero los requisitos no funcionales tales como: la disponibilidad y la confiabilidad. Se muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Se representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

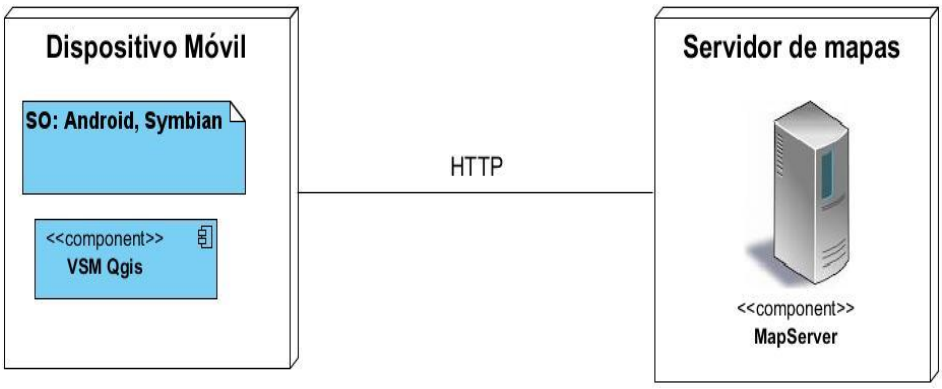

Fig. 18 Diagrama de Despliegue de VSM Qgis.

La representación del despliegue del sistema se realizó en forma de nodos. En estos nodos están recogidas las dependencias entre las instancias y sus interfaces, y también la migración de entidades entre nodos u otros contenedores.

En el nodo dispositivo móvil se encuentran la especificaciones que tiene que tener atendiendo a los sistemas operativos en los cuales será soportada la aplicación. Este nodo se conecta con el servidor

de mapas MapServer que brindará los servicios estandarizados por la OGC para que se visualicen en el dispositivo móvil.

La comunicación entre estos nodos se realiza a través de protocolos bien definidos para que la transmisión de los datos sea de manera íntegra y segura. El protocolo HTTP[9], es el establecido por la OGC basándose en la arquitectura de *Open Web Servicies* (OWS). La definición completa se denomina *Catalog Web for Servicies* (CSW). Este permite la interacción entre el dispositivo móvil con el servidor, empleando la *interface* común de pasarela CGI[10].

2.5 Conclusiones

Aplicando los conocimientos adquiridos en el desarrollo del capítulo anterior; queda diseñada la propuesta de Arquitectura de *Software* para Sistemas de Información Geográficos de escritorio basado en Qgis. Esta propuesta permite observar que la utilización del *framework* Quantum GIS, permitirá la culminación de la aplicación en un tiempo menor que si se comenzara a desarrollar desde los cimientos. La utilización de los servicios de mapas permitirá acceder a grandes volúmenes de datos sin hacer un uso inadecuado de la memoria de estos dispositivos. Se organizó lógicamente el contenido que se mostrará a los usuarios, lo que permitirá una mayor aceptación de la aplicación.

Se describen los atributos de calidad y requisitos que tienen impacto en la arquitectura, además se identifican los casos de uso arquitectónicamente significativos, donde quedan definidas las funcionalidades de la aplicación por donde comenzará el desarrollo. Se abordaron los elementos fundamentales correspondientes a las 4+1 vistas para el diseño de una correcta AS, entre los que se encuentran todos los argumentos arquitectónicamente significativos recogidos en las mismas. En la vista lógica se realizó una estructura de capas, teniendo en cuenta el estilo arquitectónico y patrones utilizados. Las vistas: implementación y despliegue muestran la infraestructura de tecnologías que se necesita para el correcto funcionamiento del SIG, haciendo énfasis en sus principales componentes. Ambas vistas muestran la distribución física del sistema, en la primera desde el punto de vista de la implementación y la segunda según los nodos en los que se despliega. Esto posibilitó una mayor comprensión de las metas y restricciones para el desarrollo de la solución. Quedando definida y documentada la arquitectura candidata. A partir de los resultados alcanzados ya se puede comenzar la validación de la propuesta de arquitectura realizada.

[9] **HTTP**: protocolo de transferencia de hipertexto, es el método más común de intercambio de información.
[10] **CGI**: Common Gateway Interface, según sus siglas en inglés.

Capítulo III

Evaluación de la Arquitectura

A continuación se realiza la evaluación de la arquitectura propuesta en el capítulo anterior. La evaluación de la arquitectura es una actividad que tiene como objetivo medir propiedades de un sistema, y de esta forma verificar si el mismo cumple con los requisitos de los clientes.

3.1 ¿Por qué evaluar una Arquitectura?

La Arquitectura de *Software* es la organización fundamental de un sistema. Esta es un producto temprano de la fase de diseño, tiene un profundo efecto en el sistema y en el proyecto, por lo que un error o una mala arquitectura pueden llevar un proyecto al fracaso, donde todos los atributos de calidad pueden quedar insatisfechos.

En el desarrollo de *software* cuanto más temprano se encuentre un problema en un proyecto es mejor. El costo de arreglar un error durante las fases de levantamiento de requisitos o en la de diseño, es mucho menor al costo de arreglar ese mismo error en la fase de pruebas. **(Evaluación de Arquitecturas de Software, 2005)** La arquitectura tiene bajo su responsabilidad decisiones que son críticas para la evolución futura de un *software* desde las disposiciones tecnológicas hasta decisiones que pueden influir en el costo del mismo. Es por estas razones que realizar una evaluación de la arquitectura es la manera más económica de evitar desastres futuros.

La arquitectura del *software* es generalmente evaluada después de que está especificada, pero antes que se comience con la implementación. La arquitectura es un proceso iterativo e incremental, por lo que puede ser evaluada al final de cada ciclo o bien puede ser evaluada en cualquier etapa de su vida. Para la evaluación existen dos variantes útiles: evaluación temprana y evaluación tardía.

Evaluación temprana: la evaluación no tiene por qué esperar a que la arquitectura esté totalmente especificada. Esta puede ser utilizada en cualquier etapa del proceso de creación de la arquitectura para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes. Por supuesto, la completitud y fidelidad de la evaluación es directamente proporcional a la completitud y fidelidad de la descripción de la arquitectura. **(Evaluación de Arquitecturas de Software, 2008)**

Evaluación tardía: esta evaluación toma lugar no solo cuando la arquitectura está terminada, también cuando la implementación está completa. Este caso ocurre cuando la organización hereda un sistema legado. La técnica para evaluar un sistema legado es la misma que para evaluar un sistema recién nacido. Una evaluación es útil para entender el sistema legado y saber si este cumple con los requerimientos de calidad y comportamiento. **(Evaluación de Arquitecturas de Software, 2008)**

La evaluación puede realizarse en cualquier momento del desarrollo, aunque es recomendable hacerlo cuando el equipo de trabajo comienza a tomar decisiones que dependen de la arquitectura. Para realizar estas evaluaciones existen técnicas y métodos que guían el proceso de evaluación.

3.2 Técnicas y Métodos para evaluar la arquitectura

La arquitectura de *software* tiene un gran impacto sobre la calidad de un sistema, por lo que es necesario tener la capacidad de tomar decisiones sobre ella. Para la evaluación de la arquitectura se conocen varias técnicas y métodos que brindan la posibilidad de escoger cuál es la arquitectura ideal para el desarrollo de la aplicación que se desee desarrollar.

Según la bibliografía consultada para el desarrollo de las técnicas de evaluación, es necesario de un esfuerzo del ingeniero de *software* para crear especificaciones y predicciones, pero teniendo en cuenta que lo que se desea es una evaluación temprana de la arquitectura, que permita la toma de decisiones de tipo arquitectónico cuando aún se está iniciando el proceso de desarrollo, en este caso es recomendable usar técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos.

En este caso Bosch **(CAMACHO, y otros, 2004)** propone diferentes técnicas de evaluación de arquitecturas de *software*: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

Las técnicas de evaluación se clasifican en cualitativas y cuantitativas. En la primera clasificación se encuentran las técnicas que son usadas cuando la arquitectura se encuentra aún en construcción y de ellas se obtienen respuestas concisas. Entre las mismas se encuentran: escenarios, cuestionarios y listas de chequeo. La segunda clasificación agrupa las técnicas que se utilizan cuando la arquitectura ya ha sido implantada. Estas son: métricas, normas, máximos y mínimos teóricos, simulaciones y prototipos.

Evaluación basada en escenarios: esta consiste en crear escenarios donde exista un contexto determinado y una respuesta, descrita a través de la arquitectura que detalla cómo debería responder el sistema.

Evaluación basada en simulación: consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. Una vez implementados estos se pueden usar un conjunto de escenarios para evaluar los atributos de calidad.

Evaluación basada en modelos matemáticos: esta técnica es usada para validar atributos de calidad operables. Este puede ser combinado con la técnica de simulación donde la entrada de uno es el resultado del otro.

Evaluación basada en experiencia: esta evaluación es propiciada por arquitectos del proyecto o externos a este. Está basada en la experiencia y la intuición de estos.

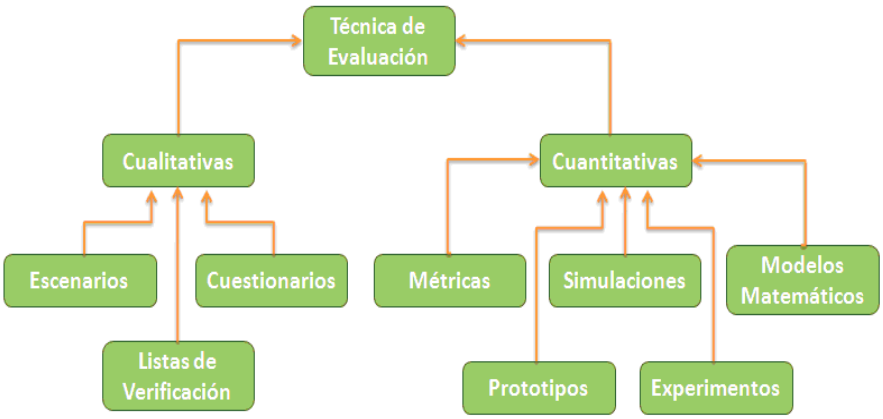

Figura 19. Técnicas de evaluación. (GestionPolis.com, 2009)

También existen definidos varios métodos que se utilizan para la evaluación como los que propone Kazman y otros autores, creadores de métodos de evaluación como el Método de Análisis de Arquitecturas de *Software* (SAAM, por sus siglas en inglés, *Software Architecture Analysis Method*), el Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) y el Método para evaluar los diseños en fases tempranas (Active Reviews for Intermediate Designs, ARID).

3.2.1 Método SAAM

El método de análisis de arquitecturas de software SAAM es uno de los primeros que fue ampliamente difundido y documentado. Originalmente fue creado para el análisis de modificabilidad de la arquitectura, pero ha demostrado ser muy útil para evaluar ciertos atributos de calidad, tales como portabilidad, escalabilidad e integrabilidad. Este consiste en la enumeración de escenarios que representarán los probables cambios a los que el sistema estará sometido en el futuro. Como entrada principal necesitará la descripción de la arquitectura de dicho sistema, la cual será evaluada. La evaluación de este método está compuesta por seis pasos, los cuales son:

Paso 1 - Desarrollar Escenarios: el primer paso en un período de sesiones SAAM, es un ejercicio de lluvia de ideas con el alcance de la identificación del tipo de actividades que el sistema tiene que soportar. Estas actividades junto con las posibles modificaciones que los interesados pueden anticipar son agrupadas en el llamado sistema de escenarios. En el desarrollo de escenarios, el desafío consiste en capturar todos los principales usos y usuarios del sistema, todos los atributos de calidad y futuros cambios en el sistema. Este ejercicio se realiza generalmente dos veces. **(Calvo San Martin, 2009)**

Paso 2 - Describir la (s) arquitectura (s): en la segunda etapa del período de sesiones SAAM se presenta la arquitectura candidata. La notaciones de la arquitectura utilizada debe ser bien entendida por los participantes y debe indicar la representación estática del sistema (componentes, sus interconexiones y la relación con el medio ambiente) así como el comportamiento dinámico del sistema. Esto puede tomar la forma de un lenguaje natural de las especificaciones de comportamiento o alguna otra especificación más formal. **(Calvo San Martin, 2009)**

Paso 3 - Clasificar y Priorizar Escenarios: en este punto el análisis de los escenarios se clasifican en escenarios directos e indirectos. Un escenario directo se apoya en la arquitectura candidata porque es basado en requisitos a partir de los cuales el sistema ha evolucionado. Los escenarios directos son perfectamente candidatos como un indicador para el rendimiento de la arquitectura o fiabilidad. Un escenario indirecto es la secuencia de eventos para que la realización o el logro de la arquitectura tengan que sufrir pequeños o grandes cambios. La priorización de los escenarios se basa en un procedimiento de votación. Puesto que SAAM es destinado a la evaluación de la modificabilidad del sistema, los resultados de la votación serán un conjunto de escenarios indirectos que se consideran más probables de ocurrir. **(Calvo San Martin, 2009)**

Paso 4 - Evaluar individualmente escenarios indirectos: en el caso de un escenario directo el arquitecto demuestra como el escenario va a ser ejecutado por la arquitectura. En el caso de un

escenario indirecto el arquitecto describe la forma en que la arquitectura necesita ser modificada para acomodar el escenario. Para cada escenario indirecto deben estar identificadas las modificaciones arquitectónicas necesarias. **(Calvo San Martin, 2009)**

Paso 5 - Evaluar la Interacción de los Escenarios: cuando dos o más escenarios están pidiendo cambios en los mismos componentes de la arquitectura, se dice que interactúan. En este caso, los componentes afectados deben ser modificados o dividido en sub-componentes, a fin de evitar la interacción de los diferentes escenarios. **(Calvo San Martin, 2009)**

Paso 6 - Crear una evaluación global: por último se le asigna un peso a cada uno de los escenarios en términos de su importancia relativa para el éxito del sistema. Sobre la base de estos escenarios puede ser propuesta una clasificación general, si se comparan varias arquitecturas. Pueden ser propuestas alternativas para la arquitectura más adecuada, cubriendo los escenarios directos que requieren menos cambios en el apoyo a los escenarios indirectos. **(Calvo San Martin, 2009)**

Dependiendo del objetivo de la evaluación será el resultado de la misma. Por ejemplo, si el objetivo es evaluar una sola arquitectura, este método enumera los lugares más vulnerables a fallos dentro de la misma, en términos de los requerimientos de modificabilidad. También puede ser para el caso de que se deseen evaluar varias arquitecturas con el fin de seleccionar una que satisfaga mejor los requerimientos de calidad y con la menor cantidad de modificaciones posibles.

3.2.2 Método ATAM

EL método de análisis de acuerdos de arquitectura ATAM está centrado en tres aéreas distintas, el estilo arquitectónico, el análisis de los atributos de calidad y el método SAAM. Su nombre surge del hecho de que revela la forma específica en que una arquitectura cumple con algunos atributos de calidad, así como la interacción de estos con otros. El método se centra en identificar el estilo arquitectónico o enfoque arquitectónico utilizado. Estos representan los métodos aplicados en la arquitectura para cumplir con los atributos de calidad**. (R. Kazmar, 2001)** La metodología de ATAM comprende nueve pasos divididos en cuatro fases:

Fase 1: Presentación

1. Presentación del ATAM: el líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.

2. Presentación de las metas del negocio: se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.

3. Presentación del negocio: el arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.

Fase 2: Investigación y análisis

4. Identificación de los enfoques arquitectónicos: estos elementos son detectados, pero no analizados.

5. Generación del *Utility Tree*: se priorizan los atributos de calidad que engloban la "utilidad" del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.

6. Análisis de los enfoques arquitectónicos: con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.

Fase 3: Pruebas

7. Tormenta o lluvia de ideas y establecimientos de la prioridad de los escenarios: con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.

8. Análisis de los enfoques arquitectónicos: este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.

Fase 4: Reportes

9. Presentación de los resultados: basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

3.2.3 Método ARID

El método de Análisis de diseños intermedios es conveniente para realizar revisiones parciales en etapas tempranas del desarrollo. Es conveniente saber si el diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. ARID es un híbrido entre *Active Design Review* (ADR)

y ATAM. ADR es utilizado para la evaluación de diseños detallados de unidades de software como los componentes o módulos. En el caso de ADR proporciona una documentación detallada del diseño y completan cuestionarios. En el caso de ATAM se realiza una evaluación de la arquitectura en su conjunto. Por esta combinación de filosofías surge ARID para la evaluación temprana de arquitecturas de *software*. El método de evaluación comprende nueve pasos divididos en dos fases:

Fase 1: Actividades previas.

1. Identificación de los encargados de la revisión: los encargados de la revisión son los ingenieros de *software* que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de encargado de revisión de ADR e involucrado del ATAM.

2. Preparar el informe de diseño: el diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.

3. Preparar los escenarios bases: el diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.

4. Preparar los materiales: se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.

Fase 2: Revisión

5. Presentación del ARID: se explican los pasos del ARID a los participantes.

6. Presentación del diseño: el líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.

7. Lluvia de ideas y establecimiento de prioridad de escenarios: se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar.

Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.

8. Aplicación de los escenarios: comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos:

- ✓ Se agota el tiempo destinado a la revisión.
- ✓ Se han estudiado los escenarios de más alta prioridad.
- ✓ El grupo se siente satisfecho con la conclusión alcanzada.

Puede suceder que el diseño presentado sea conveniente, con la exitosa aplicación de los escenarios, o por el contrario, no conveniente, cuando el grupo encuentra problemas o deficiencias.

9. Resumen: al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

No se puede decir que un método es mejor que otro, en cambio sí se puede afirmar que en determinadas condiciones y para determinados atributos de calidad existen métodos que evalúan la arquitectura de manera más eficiente que otros.

Para la evaluación de la propuesta que se realiza en esta investigación se empleará el método SAAM. Este permite la evaluación temprana de la arquitectura, por medio de escenarios de los atributos de calidad que demanda la arquitectura. Este método ha demostrado ser muy eficiente para evaluar muchos atributos de calidad rápidamente.

3.3 Evaluación de la arquitectura propuesta utilizando el método SAAM

Paso 1. Desarrollo de escenarios

Los escenarios identificados para esta herramienta son los que se exponen a continuación, además de los atributos de calidad que se ponen de manifiesto:

A) Un usuario desea realizar análisis sobre el mapa y la aplicación debe permitir realizar cálculos de distancias, identificar o localizar objetos. (Funcionalidad)

B) Un usuario desea visualizar el mapa en diferentes tamaños y regiones, entonces la aplicación debe permitir realizar la navegación. (Funcionalidad)

C) Un usuario pide ayuda en un contexto determinado y la aplicación debe de ser capaz de ofrecer ayuda para ese contexto. (Usabilidad)

D) Un usuario desea cambiar su interfaz de la aplicación y el sistema debe mostrar la nueva interfaz. (Modificabilidad)

E) Un usuario desea cambiar el sistema operativo de su móvil de Android a Symbian o viceversa y la aplicación debe poder instalarse o ejecutarse en los sistemas operativos Symbian o Android. (Portabilidad)

F) Un usuario desea agregar nuevas funcionalidades a la aplicación y esta debe de contar con una estructura que permita el incremento de nuevos complementos. (Escalabilidad)

G) Un usuario desea interactuar con el sistema mientras está en movimiento y la aplicación debe establecer la conexión con el servidor de mapas desde cualquier lugar en todo momento (Portabilidad) (Disponibilidad)

H) Un número significativo de usuarios (100) acceden en el mismo instante de tiempo al servidor de mapas y este debe de contar con los mecanismos que le permitan atender todas las peticiones que le realizan en el menor tiempo posible. (Eficiencia)

I) El administrador desea actualizar o modificar el servidor de mapas y la aplicación debe de tener los mecanismos que le permitan conectarse al nuevo servidor. (Modificabilidad)

J) Un usuario producto a acciones externas o internas le falla el sistema, este debe ser capaz de estar disponible en 24 horas. (Confiabilidad)

K) Un usuario desea actualizar la aplicación y esta deber tener creados los mecanismos para realizar la actualización. (Portabilidad)

L) Un usuario desea agregar nuevos puntos y rectas al mapa, el sistema debe permitir la edición de las capas del mapa que el cliente desee. (Funcionalidad)

M) Un usuario debe trabajar en una zona de silencio, en un bosque o dentro de un túnel. El sistema debe de permitir obtener la cantidad de datos suficientes para el trabajar independiente del servidor por un periodo de tiempo. (Disponibilidad)

N) El usuario recibe una llamada en el momento en que está trabajando, el sistema debe ser capaz de mantener la integridad de los datos con lo que estaba trabajando. (Confiabilidad)

Paso 2. Descripción de la Arquitectura

Para realizar la propuesta de arquitectura se tienen en cuenta las características y las tecnologías disponibles para el desarrollo de la aplicación. Por lo que se propone aplicar los estilos arquitectónicos orientado a objetos, por capas y el cliente/servidor, esto es porque las tareas van a estar divididas entre los dispositivos móviles y los servidores. Los patrones utilizados responden a los que se identificaron de la herramienta Qgis y los que van a respaldar al estilo seleccionado. Lo que queda como propuesta de desarrollo es una aplicación nativa de dispositivos móviles basada en el *framework* Qgis, multiplataforma, que permitirá representar un Sistema de Información Geográfica de escritorio en los *Smartphone*.

Paso 3. Clasificación y priorización de los escenarios

Escenario	Clasificación	Atributo	Priorizados
A	Directo	Funcionalidad	
B	Directo	Funcionalidad	
C	Directo	Usabilidad	
D	Indirecto	Modificabilidad	X
E	Directo	Portabilidad	
F	Directo	Escalabilidad	
G	Directo	Portabilidad y Disponibilidad	
H	Directo	Eficiencia	
I	Indirecto	Modificabilidad	
J	Directo	Confiabilidad	
K	Directo	Portabilidad	
L	Directo	Funcionalidad	
M	Indirecto	Disponibilidad	X

N	Indirecto	Confiabilidad	X

Tabla. 4 Clasificación de los escenarios.

Se han clasificado los escenarios según el apoyo que le brinda a la arquitectura candidata. De los escenarios identificados se cuenta con 10 escenarios directos y 4 indirectos. Los escenario que según el equipo de revisión piensa que tienen más posibilidades de ocurrir son el D, M y N, por lo que priorizan estos como se muestra en la tabla anterior.

Paso 4. Evaluación de escenarios

Escenarios Indirectos	**Acciones**
D	Se debe incorporar un nuevo componente que le brinde la posibilidad al usuario de personalizar su interfaz.
I	Se debe agregar un nuevo componente que identifique, si existen actualizaciones o modificaciones en el servidor y teniendo esta información establecer la comunicación.
M	Se debe incorporar un mecanismo para adquirir suficiente información para trabajar independientemente del servidor durante un período de tiempo.
N	El sistema debería optimizarse para que no exista la pérdida de datos y la recuperación de fallos.

Tabla. 5 Evaluación de escenarios.

En este epígrafe quedan listados los escenarios indirectos y las decisiones arquitectónicas que se deben tomar para modificar y dar solución a las situaciones por la cual se ven afectados. No se han mostrado los escenarios directos, pues a estos la arquitectura le da respuesta sin necesidad de modificarse.

Paso 5. Interacción de escenarios

Los escenarios indirectos no interactúan sobre ningún componente. Se muestra que existe alta cohesión en la arquitectura descrita.

Paso 6. Evaluación general

Con el objetivo de asignarle un peso a los escenarios para evaluar la importancia relativa para el éxito del negocio. Se establece que el rango de puntuación estará establecido en un orden de 1…5, donde 5 representa la mayor importancia.

Escenario	Clasificación	Atributo	Puntuación
A	Directo	Funcionalidad	5
B	Directo	Funcionalidad	5
C	Directo	Usabilidad	2
D	Indirecto	Modificabilidad	1
E	Directo	Portabilidad	4
F	Directo	Escalabilidad	4
G	Directo	Portabilidad y Disponibilidad	5
H	Directo	Eficiencia	3
I	Indirecto	Modificabilidad	2
J	Directo	Confiabilidad	3
K	Directo	Portabilidad	4
L	Directo	Funcionalidad	5
M	Indirecto	Disponibilidad	3
N	Indirecto	Confiabilidad	3

Tabla. 6 Evaluación general de los escenarios.

Después de haber completado los 6 pasos propuestos por el método de evaluación escogido, aplicando la técnica utilizada por él. Con el fin de obtener posibles problemas en el diseño de la aplicación, que puedan provocar un aumento del costo de producción. Se puede decir que se han identificado escenarios que provocarán cambios en la herramienta, se han priorizados los que tienen más posibilidades de ocurrir a consideración del equipo de evaluación. Se describen los que tienen mayor importancia para la evolución de la propuesta.

3.4 Conclusiones

El estudio de las técnicas y métodos para la evaluación de la arquitectura, sentaron las bases para evaluar la propuesta que se realiza con el fin de encontrar errores en el diseño arquitectónico. Al ser detectados estos antes de comenzar la implementación permite refinar la arquitectura para hacerla más robusta.

La evaluación de la arquitectura propuesta permitió que se identificaran los escenarios los indirectos, que provocarán modificaciones en el sistema. Se priorizaron los escenarios con más posibilidades de ocurrir, debido a que estas son metas que debe cumplir la propuesta arquitectónica. Esto permite que se minimicen los riesgos y se tomen decisiones que favorezcan el desarrollo de la herramienta. El diseño de la arquitectura propuesta presenta como característica que es funcional, portable y escalable. Se puede decir que es la adecuada para el desarrollo de la aplicación.

Conclusiones Generales

En el presente trabajo se realizó un estudio de los principales aspectos de la descripción arquitectónica y las tendencias de desarrollo en los dispositivos móviles, seleccionándose el estilo arquitectónico adecuado para la solución que forma parte del núcleo de la arquitectura. Se realizó un estudio profundo de técnicas y métodos para la evaluación de la arquitectura, donde se propuso el método SAAM para evaluar la misma de manera temprana.

Una vez culminada esta investigación, quedando como resultado principal, el diseño de la propuesta de arquitectura para dispositivos móviles basado en el *framework* Quantum GIS, para darle solución a los objetivos de la misma. Se puede concluir que:

- ✓ El desarrollo para los dispositivos móviles es muy heterogéneo, producto a los dispositivos para los que se desarrolla, el tamaño de la pantalla que estos presentan, además del entorno de trabajo de los mismos.
- ✓ La utilización de Quantum GIS como base para el desarrollo, permitirá reducir el tiempo de programación y será una ventaja para la aplicación.
- ✓ El hecho de que se desarrolle sobre tecnología libre es una buena opción, más allá de los beneficios económicos que representa, ya que permite que le desarrollador aproveche el *software* al máximo.
- ✓ Se definió y documentó la arquitectura candidata enfocada a lograr desarrollar un Sistema de Información Geográfica de escritorio para dispositivos móviles basados en las bibliotecas que ofrece Quantum GIS.
- ✓ En base a los atributos de calidad que se propusieron alcanzar, la arquitectura propuesta puede ser aplicada en el desarrollo de SIG de escritorio para dispositivos móviles con SO Symbian o Android.
- ✓ La evaluación de la arquitectura mostró el cumplimiento de los objetivos de la presente investigación, por la correspondencia entre los requisitos y atributos de calidad. El diseño de la arquitectura propuesta anticipa que se obtendrá una aplicación funcional, de gran portabilidad y escalabilidad.

Recomendaciones

De acuerdo al trabajo realizado y a la importancia conferida al mismo. Con el fin que se continúe mejorando la propuesta realizada se recomienda:

- ✓ Realizar una evaluación tardía de la arquitectura propuesta aplicando otro de los métodos estudiados como lo es el ATAM.
- ✓ Analizar y aplicar refinamientos a la arquitectura propuesta durante el ciclo de desarrollo.
- ✓ Analizar la posibilidad de expandir la aplicación hacia otras plataformas de los dispositivos móviles.
- ✓ Aplicar la propuesta de arquitectura para el futuro desarrollo de Sistemas de Información Geográfica para dispositivos móviles.
- ✓ Implementar los casos de uso arquitectónicamente significativos en la primera iteración de software.

Referencias Bibliográficas

Gianfelici, Esteban. 2008. Mapas y Mapas.com.ar. *Mapas y Mapas.com.ar.* [En línea] Universidad Nacional del Litoral, Argentina, julio de 2008. [Citado el: 11 de mayo de 2012.] http://www.mapasymapas.com.ar/que%20es%20un%20SIG.php.

Dispositivos Móviles. **Gevara Soriano, Anaid. 2010.** 07, México : s.n., 2010.

DT-Arquitectura de la UCi. 2010. *Vista de Arquitectura de Integración.* 2010.

Evaluación de Arquitecturas de Software. **Bustacara Medina, Cesar Julio. 2008.** 2008.

Kruchten, Philippe . 2006. *Planos Arquitectónicos: El Modelo de "4+1" Vistas de la Arquitectura del Software.* 2006.

Rodríguez Fuentes, Lewis. 2011. *Desarrollo de paquete de Arquitecturas de Software para Sistemas de Información Geográfica de escritorio.* La Habana : s.n., 2011.

A field guide to Boxology: Preliminary classification of architectural styles for software systems. **Shaw, Mary y Clements, Paul. 1996.** s.l. : Proceedings of the 21st International Computer Software and Applications Conference, 1996.

A Survey of Architecture Description Languages. **Clements, Paul C. 1996.** Washington, DC : the Association for Computing Machinery, 1996.

Arquitectura de Software. **CAMACHO, ERIKA. 2004.** 2004.

D. Nielsen, Paul. Sotfware Engineering Institute. [En línea] http://www.sei.cmu.edu/architecture/start/glossary/community.cfm.

Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. **Reynoso , Carlos y Kicillof, Nicolás . 2004.** BUENOS AIRES : s.n., 2004.

—. **Reynoso, Carlos Billy y Kicillof, Nicolás. 2004.** 2004.

Hernández Cervante, Beatriz y Calderio Hechevarria, Dairon Freddy. 2009. Diseño y evaluación de la Arquitectura de Software con la tecnología J2EE y el Framework Spring. La Habana : s.n., 2009.

IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. **IEEE. 2000.** 2000.

Introducción a la Arquitectura de Software. **Reynoso, Carlos Billy. 2004.** Buenos Aires : s.n., 2004.

Proceedings of the Conference on The Future of Software Engineering. **Garlan, David. 2000.** New York : ACM, 2000.

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

Software architecture: a roadmap. **Garlan, David. 2000.** New York, NY : the Association for Computing Machinery, 2000.

Díaz Salvador, Sergi . 2010. *Diseño y desarrollo de una aplicación en ArcGIS Mobile.* 2010.

Free Software Foundation. 2001. Sistema Operativo GNU. *Sistema Operativo GNU.* [En línea] Free Software Foundation, 2001. [Citado el: 26 de 04 de 2012.] http://www.gnu.org/philosophy/free-sw.es.html.

2009. GestionPolis.com. *GestionPolis.com.* [En línea] 07 de 05 de 2009. [Citado el: 30 de mayo de 2012.] http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

GUERRERO, ALEJANDRO SALAZAR. 2010. *TECNOLOGÍAS MÓVILES.* 2010.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. Fases dentro de un ciclo. [aut. libro] Grady Booch, James Rumbaugh Ivar Jacobson. *El proceso unificado de desarrollo de software.* Madrid : Pearson educación. S.A, 2000.

—. 2000. La vida del proceso unificado. [aut. libro] Grady Booch, James Rumbaugh Ivar Jacobson. *El proceso unificado de desarrollo de software.* Madrid : Pearson Educación S.A, 2000.

Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). **Letelier , Patricio y Penadés, Mª Carmen . 2004.** Valencia : s.n., 2004.

Montesinos Lajara, Miguel y Carrasco Marimon, Javier. *gvSIG Mobile, GvSIG para dispositivos móviles.*

Sistema y recursos para tener éxito en internet. 2012. masadelante.com. *masadelante.com.* [En línea] 2012. [Citado el: 22 de 05 de 2012.] http://www.masadelante.com/faqs/sistema-operativo.

Anexos

Anexo 1: Descripción entre los sistemas operativos de dispositivos móviles.

Symbian OS

Es un sistema operativo abierto y estándar para dispositivos de telefonía móvil. Está licenciado por los principales desarrolladores de telefonía móvil del mundo, tales como: Motorola, Nokia, Samsung, Sony Ericsson. Symbian posee ciertas características que influyen de manera determinante en el desarrollo de aplicaciones, ellas son:

Está basado en un micro kernel, es decir, una mínima porción del sistema tiene privilegios de kernel, el resto se ejecuta con privilegios de usuario, en modo de servidores.

Cada aplicación corre en sus propios procesos y tiene acceso solo a su propio espacio de memoria. Esto permite que las aplicaciones para Symbian sean orientadas a "*single threads*" y no múltiples.

El sistema posee componentes que permiten el diseño de aplicaciones multiplataforma, o sea, diferentes tamaños de pantalla, color, resolución, teclados. La mayoría de estos componentes han sido diseñados en C++.

Manejo fiable de los datos, incluso en caso de fallo en la comunicación o falta de recursos, como memoria, disco o batería.

Este es un sistema operativo que presenta una arquitectura 3 capas.

Android

Es un Sistema Operativo además de una plataforma de *software* basada en el núcleo de Linux. Diseñada en un principio para dispositivos móviles, Android permite controlar dispositivos por medio de bibliotecas desarrolladas o adaptadas por Google mediante el lenguaje de programación Java. Es una plataforma de código abierto. Entre sus características presenta:

Framework de aplicaciones: permite el reemplazo y la reutilización de los componentes.

SQlite: base de datos para almacenamiento estructurado que se integra directamente con las aplicaciones.

Windows Mobile

Arquitectura modular

Windows CE 6.0 está desarrollado como un conjunto de módulos independientes. Su arquitectura le permite al fabricante que elija implantar este sistema operativo en su dispositivo, el placer de personalizarlo a su gusto y riesgo, ya que Microsoft solo garantiza un número limitado de versiones probadas, que coinciden con las diferentes versiones comerciales del sistema operativo Windows CE 6.0 y que aún no han salido al mercado. Aun así, la limitación es nula, pues además de ofrecer una plataforma de desarrollo del sistema operativo y un entorno de programación para desarrollar las aplicaciones para Windows CE 6.0, en la última versión de Windows CE 6.0 Microsoft ofrece una licencia *shared code* que proporciona a su propietario el código fuente del sistema operativo.

De momento Microsoft ha alcanzado acuerdos comerciales con las siguientes compañías para que monten Windows CE 6.0 en sus dispositivos: AT&T, Chunghwa Telecom, Dopod International Corp., Orange, HP, HTC, iMate, LG Electronics, Motorola Inc., Palm Inc., Samsung, en Sprint, Telefónica, Toshiba, Verizon Wireless y Vodafone.

IOS

Objective-C es un lenguaje de programación orientado a objetos creado como un súper conjunto de C pero que implementase un modelo de objetos parecido al de *Smalltalk*. Se usa como lenguaje principal de programación en Mac OS X y GNUstep.

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con Mac OS X. Trabaja conjuntamente con Interface *Builder*, una herencia de *NeXT*, una herramienta gráfica para la creación de interfaces de usuario. Incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Objective-C, Objective-C++, Java y *AppleScript* mediante una amplia gama de modelos de programación.

La arquitectura iOS está basada en capas, donde las capas más altas contienen los servicios y tecnologías más importantes para el desarrollo de aplicaciones, y las capas más bajas controlan los servicios básicos.

Anexo 2. Ejemplos de Sistemas de Información Geográfica

GvSIG Mobile

Es una herramienta que dentro de sus funcionalidades accede a datos vectoriales en formato SHP, KML (Keyhole Markup Language), GML (Lenguaje de Marcado Geográfico) y GPX *eXchange Format* (Formato de Intercambio GPS). Es capaz de acceder a datos ráster en local ECW, JPEG, PNG, GIF y a servicios remotos como el WMS (Web Map Service). Posee herramientas de navegación, de consulta, para la obtención de información mediante búsquedas alfanuméricas, mediciones, selecciones por objetos, edición de las propiedades de las capas. La funcionalidad que le convierte en un potente cliente móvil de datos SIG es la edición gráfica de datos vectoriales utilizando GPS o no y la edición alfanumérica de estos datos mediante la utilización de formularios personalizados. Para la realización de estos formularios se ha utilizado la herramienta *Thinlet*, que permite definir dichos formularios de entrada de datos utilizando ficheros XML. La interfaz de usuario cobra especial importancia para GvSIG Mobile. Se ha desarrollado un sistema de barras de herramientas en el que se incorpora el concepto de grupo de barras de herramientas que permite ocultar botones y dejar libre más espacio para los mapas, así como un sistema de ayuda contextual y controles semitransparentes con el mismo objetivo.

GvSIG Mini

Es un SIG que está diseñado para dispositivos móviles que presentan Java o Android y que permite la visualización y navegación sobre cartografía digital. Es desarrollado por la empresa Prodevelop. La arquitectura está basada en una arquitectura multihilo. La construcción de la interfaz de usuario se llevó a cabo utilizando el *Lightweight UI Toolkit,* que es un *framework* desarrollado por SUN*,* licenciado como GPL para la creación de interfaces de usuario que se comporten igual en todos los dispositivos, de forma análoga a *Swing* en Java SE.

ArcGIS Mobile

En esta herramienta los proyectos móviles y los datos cartográficos se gestionan e implementan de forma centralizada con ArcGIS *Server*. Incluye una aplicación móvil orientada a tareas y lista para su uso para recopilar datos de SIG, inspeccionar y representar cartográficamente. ArcGIS Mobile incluye un conjunto de bibliotecas para desarrolladores con rudimentos para crear aplicaciones móviles personalizadas con Microsoft Visual Studio .NET. El SDK[11] incluye además un conjunto de componentes para desarrolladores, documentación y muestras de código, que se pueden emplear

[11] Kit de desarrollo de software (siglas en inglés de *software development kit*).

para crear aplicaciones especializadas, que funcionen sobre una serie de distintos dispositivos (como pueden ser un teléfono inteligente Windows Mobile Smartphone, Windows Mobile Pocket PC, Windows CE .NET y Windows XP y Vista).

Quantum GIS

Esta aplicación es un Sistema de Información Geográfica que es multiplataforma. Fue uno de los primeros proyectos de la Fundación OSGeo[12]. Es una aplicación de código abierto, además soporta el trabajo con las extensiones espaciales de PostgreSQL y PostGIS. Esta herramienta es desarrollada con QT y C++. Entre sus funcionalidades fundamentales se encuentran; manejar formatos ráster y vectoriales a través de las bibliotecas GDAL[13] y OGR[14], así como bases de datos. Este SIG no presenta una arquitectura definida donde se represente la relación de sus componentes, aunque en el curso académico pasado se presentó una tesis de grado donde se propuso un paquete de arquitectura Orientado a Objetos (OO) y basado en componentes para esta herramienta.

Anexo 3: Familias de Estilos

Estilos de Flujos de Datos: esta familia es generalmente utilizada cuando se desea la reutilización y la modificabilidad de la aplicación en sistemas que presentan transformaciones; como ejemplo de estas se pueden encontrar las arquitecturas de proceso secuencial y la de tubería y filtros.

Tubería y filtros: esta arquitectura en esencia consiste en transformar un flujo de datos en un proceso comprendido por varias secciones secuenciales, donde la salida de cada uno sea la entrada del siguiente.

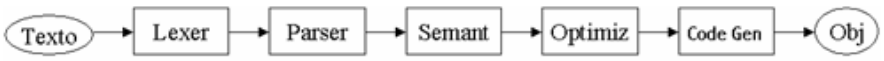

Fig. 20 Compilador en Tubería y Filtro.

El estilo tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. En el estilo

[12] ***Fundación OSGeo**: Es la fundación para el código abierto geoespacial, es una organización sin ánimos de lucro cuyo objetivo principal es apoyar y promocionar el desarrollo abierto y colaborativo de los datos y las tecnologías geoespaciales.*

[13] **Geospatial Data Abstraction Library** o GDAL es una biblioteca de software para la lectura y escritura de formatos de datos geoespaciales.

[14] **OGR**: Soporte para capas vectoriales y ráster.

secuencial por lotes los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente. **(Estilos y Patrones en la Estrategia de Arquitectura de Microsoft, 2004)**

Estilos de Llamada y Retorno: por lo general los miembros de esta familia son los más generalizados en el momento de implementar aplicaciones de gran escala, siendo estilos que enfatizan en la modificabilidad y escalabilidad.

Modelo-Vista-Controlador (MVC): separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes. El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

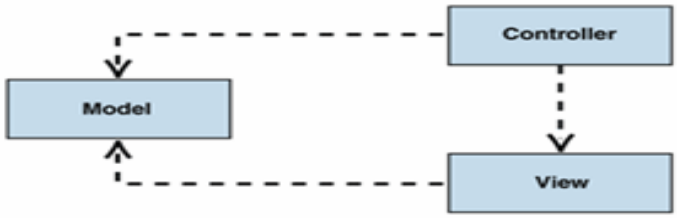

Fig. 21 Modelo-Vista-Controlador.

Arquitecturas en Capas: Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

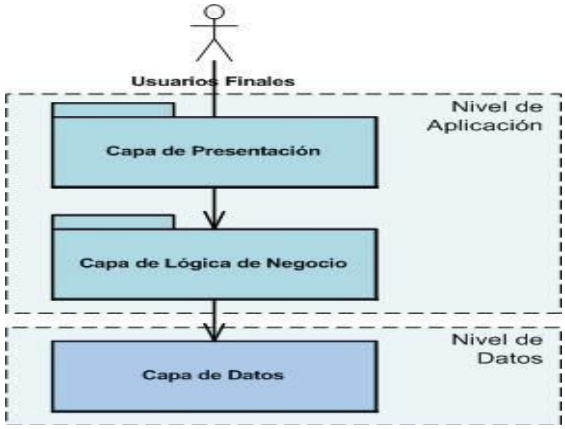

Fig.22 Arquitectura en Capas.

El estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. El estilo admite muy naturalmente optimizaciones y refinamientos y también proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. **(Reynoso, y otros, 2004)**.

Arquitecturas Orientadas a Objetos: las arquitecturas orientadas a objetos han sido clasificadas de formas diferentes, conforme a los diferentes puntos de vista que alternativamente enfatizan la jerarquía de componentes, su distribución topológica o las variedades de conectores. Los objetos representan una clase de componentes que ellos llaman *managers*, debido a que son responsables de preservar la integridad de su propia representación **(Reynoso, y otros, 2004)**. Entre sus principales ventajas se puede mencionar que permite modificar la implementación de un objeto sin afectar a sus clientes. Asimismo es posible descomponer problemas en colecciones de agentes en interacción. Además, un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Arquitecturas Basadas en Componentes: el objetivo de la arquitectura orientada a componentes es construir aplicaciones complejas mediante ensamblado de componentes que han sido previamente diseñados por otros desarrolladores o por subgrupos del equipo de desarrollo, a fin de ser rehusados en múltiples aplicaciones. En la arquitectura orientada a componentes la interfaz constituye el elemento básico de conexión. Cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. Además, debe operar con independencia de los mecanismos internos que utilice para soportar la funcionalidad de la interfaz.. **(Reynoso, y otros, 2004)**.

Estilos Peer – to – Peer: enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades pero no controlarlas directamente.

Arquitectura Orientada a Servicios (SOA): sólo recientemente esta arquitectura que los conocedores llaman SOA han recibido tratamiento intensivo en el campo de exploración de los estilos. Al mismo tiempo se percibe una tendencia a promoverlas de un sub-estilo propio de las configuraciones distribuidas que eran anteriormente a un estilo en plenitud. La visualizan como la tendencia que habrá

de ser dominante en la primera década del nuevo milenio. Este predominio no se funda en la idea de servicios en general, comunicados de cualquier manera, sino que más específicamente va de la mano de la expansión de los Web *services* basados en XML, en los cuales los formatos de intercambio se basan en XML 1.0 *Namespaces* y el protocolo de elección es SOAP. Este protocolo significa un formato de mensajes que es XML, comunicado sobre un transporte que por defecto es HTTP. **(Estilos y Patrones en la Estrategia de Arquitectura de Microsoft, 2004)**

Anexo 4: Patrones de diseño.

Patrones GRAPS

Polimorfismo: siempre que se tenga que llevar a cabo una responsabilidad que dependa del tipo.

Bajo Acoplamiento: el acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, las conoce y recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases. Acoplamiento alto significa que una clase recurre a muchas otras clases. Esto presenta los siguientes problemas: los cambios de las clases afines ocasionan cambios locales, difíciles de entender cuando están aisladas y difíciles de reutilizar puesto que dependen de otras clases.

Alta Cohesión: la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una baja cohesión hace tareas no afines o realiza trabajo excesivo.

Patrones GoF

Tipo de Patrón	Características

Creacionales	**Object Pool** (Conjunto de Objetos): se obtienen objetos nuevos a través de la clonación. Utilizado cuando el costo de crear una clase es mayor que el de clonarla. Especialmente con objetos muy complejos. Se especifica un tipo de objeto a crear y se utiliza una interfaz del prototipo para crear un nuevo objeto por clonación. El proceso de clonación se inicia instanciando un tipo de objeto de la clase que queremos clonar. **Abstract Factory** (Fábrica abstracta): permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. **Builder** (Constructor virtual): abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto. **Prototype** (Prototipo): crea nuevos objetos clonándolos de una instancia ya existente. **Singleton** (Instancia única): garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.
Estructurales	**Composite** (Objeto compuesto): permite tratar objetos compuestos como si se tratase de uno simple. **Decorator** (Envoltorio): añade funcionalidad a una clase dinámicamente. **Flyweight** (Peso ligero): reduce la redundancia cuando gran cantidad de objetos poseen idéntica información. **Proxy**: mantiene un representante de un objeto.
Comportamiento	**Command** (Orden): encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma. **Interpreter** (Intérprete): dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo. **Iterator** (Iterador): permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos. **Mediator** (Mediador): define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto. **Memento** (Recuerdo): permite volver a estados anteriores del sistema. **State** (Estado): permite que un objeto modifique su comportamiento cada vez

	que cambie su estado interno. **Strategy** (Estrategia): permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución. **Template Method** (Método plantilla): define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura. **Visitor** (Visitante): permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

Tabla 7: Patrones de diseño Gof.

Anexo 5: Lenguajes de Descripción de la Arquitectura

ADL	Investigador (Organización)	Observaciones
Acme	Monoe y Garlan (Carnegie Mellon University, CMU), David S. Wile (University of Southern California, USC)	Lenguaje de intercambio de ADLs
Aesop	Garlan (Carnegie Mellon University, CMU)	ADL de propósito general, hace énfasis en los estilos
ArTek (ARDEC/Teknowledge)	Terry, Hayes-Roth, Erman (Dominio de Arquitectura de Aplicaciones Específicas, DSSA)	Lenguaje específico de dominio. Algunos autores no lo califican como ADL
Armani	Monroe (Carnegie Mellon University, CMU)	ADL asociado a Acme
C2 SADL	Taylor, Medvidovic (University of California, Irvine , UCI)	ADL específico de estilo
CHAM	Berry, Boudol	Lenguaje de especificación

Darwin	Magee, Dulay, Eisenbach, Kramer	ADL con énfasis en dinámica
Jacal	Kicillof, Yankelevich (Universidad de Buenos Aires)	Adl- Notación de alto nivel para descripción y prototipado
LILEANNA	Tracz (Loral Federal)	Lenguaje de conexión de módulos
MetaH	Binns, Englehart (Honeywell)	ADL específico de dominio
Rapide	Luckharn (Stanford)	ADL y simulación
SADL (Structural Architecture Description Language)	Moriconi, Riemenschneider (SRI)	ADL con énfasis en mapeo de refinamiento
UML	Rumbaugh, Jacobson, Booch (Rational)	Lenguaje unificado de modelado genérico. Algunos autores no lo califican como ADL
UniCon	Shaw (Carnegie Mellon University, CMU)	ADL de propósito general, énfasis en conectores y estilos.
Wright	Garlan (Carnegie Mellon University, CMU)	ADL de propósito general, énfasis en comunicación
xADL	Medvidovic, Taylor (University of California Irvine, UCI)	ADL basado en XML

Tabla 8: Lenguajes de Descripción Arquitectónica. (Reynoso, y otros, 2004)

Anexo 6: Vista de Implementación

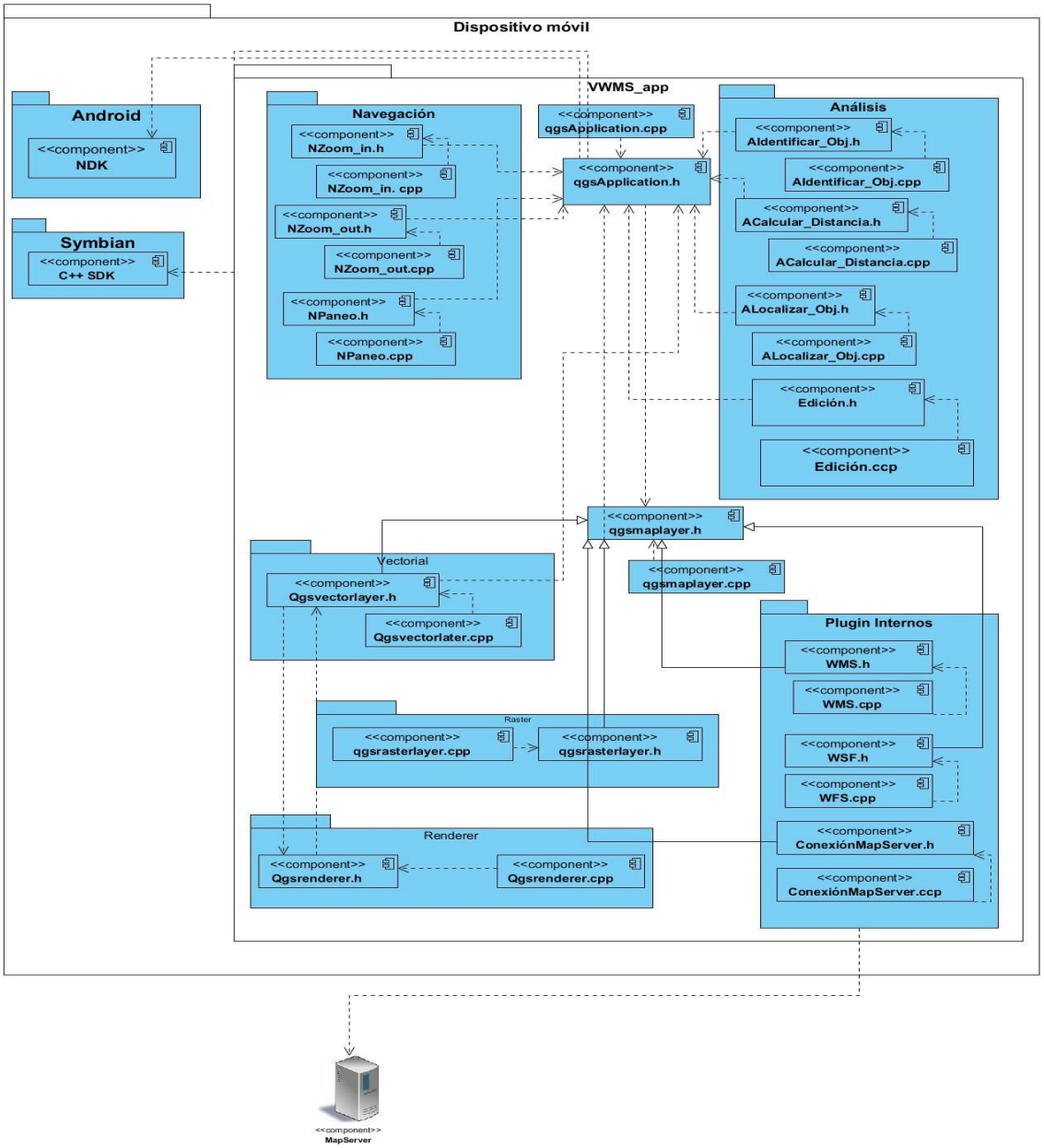

Fig. 23 Vista de Implementación

Glosario de Términos

Aplicación: programa informático que permite a un usuario utilizar una computadora con un fin específico.

Abstracción: es la capacidad de identificar o seleccionar los elementos esenciales de un sistema.

Atributos de calidad: son lo que definen la calidad y las características debe tener un sistema.

Arquitectura de Software: es la organización de un sistema donde se representan sus componentes y las relaciones entre estos.

Arquitectura de Información: encargada de diseñar, organizar, distribuir la información de una aplicación.

ADL: Architecture Description Language, según sus siglas en inglés, en español es lenguaje de descripción de la arquitectura. Deben proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones.

Componentes: representan los elementos computacionales primarios de un sistema. En la mayoría de los ADLs los componentes pueden exponer varias interfaces, las cuales definen puntos de interacción entre un componente y su entorno.

Conectores: representan interacciones entre componentes. Los conectores también tienen una especie de interfaz que define los roles entre los componentes participantes en la interacción.

Dispositivos inalámbricos: son los dispositivos que pueden comunicarse o conectarse a una red sin necesidad de cableado.

Estilos: representan familias de sistemas, un vocabulario de tipos de elementos de diseño y de reglas para componerlos.

Gdal: es una biblioteca de software para la lectura y escritura de formatos de datos geoespaciales.

OGC: *Open Geospatial Consortium* en español Consorcio Geoespacial Abierto. Su fin es la definición de estándares abiertos e interoperables dentro de los Sistemas de Información Geográfica y persigue

acuerdos que posibiliten la interoperación de sus sistemas de geoprocesamiento y faciliten el intercambio de la información geográfica en beneficio de los usuarios.

GPL: *General Public License*, licencia pública general de GNU/Linux está principalmente orientada a proteger la libre distribución, modificación y uso del software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

OGR: soporte para capas vectoriales y ráster.

OsGEO: es la fundación para el código abierto geoespacial, es una organización sin ánimos de lucro cuyo objetivo principal es apoyar y promocionar el desarrollo abierto y colaborativo de los datos y las tecnologías geoespaciales.

SDK: Kit de Desarrollo de Software (siglas en inglés de software development kit).

SIG: Sistema de Información Geográfica.

Sistema: es el conjunto de partes interrelacionadas que permiten almacenar y procesar la información.