



Universidad de San Carlos de Guatemala  
Facultad de Ingeniería  
Escuela de Ingeniería en Ciencias y Sistemas

## **DEFINICIÓN DE LA ARQUITECTURA SKRN**

**Oscar Estuardo de la Mora Hernández**

Asesorado por el Ing. José Ricardo Morales Prado

Guatemala, octubre de 2013

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**DEFINICIÓN DE LA ARQUITECTURA SKRN**

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA  
FACULTAD DE INGENIERÍA

POR

**OSCAR ESTUARDO DE LA MORA HERNÁNDEZ**

ASESORADO POR EL ING. JOSÉ RICARDO MORALES PRADO

AL CONFERÍRSELE EL TÍTULO DE

**INGENIERO EN CIENCIAS Y SISTEMAS**

GUATEMALA, SEPTIEMBRE DE 2013

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA  
FACULTAD DE INGENIERÍA



**NÓMINA DE JUNTA DIRECTIVA**

DECANO	Ing. Murphy Olympto Paiz Recinos
VOCAL I	Ing. Alfredo Enrique Beber Aceituno
VOCAL II	Ing. Pedro Antonio Aguilar Polanco
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Walter Rafael Véliz Muñoz
VOCAL V	Br. Sergio Alejandro Donis Soto
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

**TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO**

DECANO	Ing. Murphy Olympto Paiz Recinos
EXAMINADOR	Ing. Juan Álvaro Díaz Ardavín
EXAMINADOR	Ing. José Ricardo Morales Prado
EXAMINADOR	Ing. Ludving Federico Altan Sac
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

## **HONORABLE TRIBUNAL EXAMINADOR**

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

### **DEFINICIÓN DE LA ARQUITECTURA SKRN**

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 6 de junio de 2013.



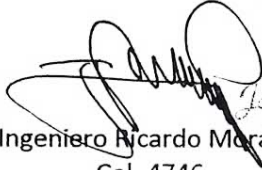
**Oscar Estuardo de la Mora Hernández**

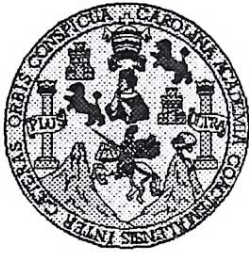
Guatemala 31 de agosto de 2013

Ingeniero  
Carlos Azurdía  
Presente

El motivo de la presente es para informarles que yo Ing. Ricardo Morales, colegiado número 4746, asesor de tesis del estudiante Oscar Estuardo de la Mora Hernández, identificado con el carnet 200412917, estudiante de Ingeniería en Ciencias y Sistemas, habiéndolo supervisado en la realización de su trabajo de graduación con tema "Definición de la Arquitectura SKRN" y realizado las correcciones correspondientes, doy por aprobada la redacción final del documento, para su posterior entrega a las respectivas unidades de la universidad.

Agradeciendo de antemano la atención de esta misiva, se despide de usted, atentamente.

  
Ingeniero Ricardo Morales  
Col. 4746  
Ricardo Morales Prado  
INGENIERO EN SISTEMAS  
COLEGIADO No. 4746



Universidad San Carlos de Guatemala  
Facultad de Ingeniería  
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 11 de Septiembre de 2013

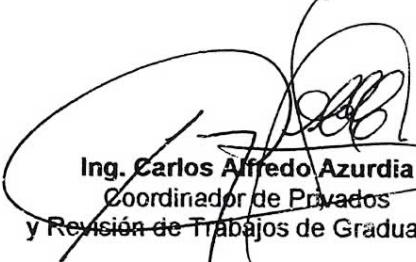
Ingeniero  
**Marlon Antonio Pérez Turk**  
Director de la Escuela de Ingeniería  
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **OSCAR ESTUARDO DE LA MORA HERNÁNDEZ** carné **2004-12917**, titulado: **"DEFINICIÓN DE LA ARQUITECTURA SKRN"**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

  
**Ing. Carlos Alfredo Azurdia**  
Coordinador de Privados  
y Revisión de Trabajos de Graduación



UNIVERSIDAD DE SAN CARLOS  
DE GUATEMALA



FACULTAD DE INGENIERÍA  
ESCUELA DE CIENCIAS Y SISTEMAS  
TEL: 24767644

E  
S  
C  
U  
E  
L  
A

D  
E

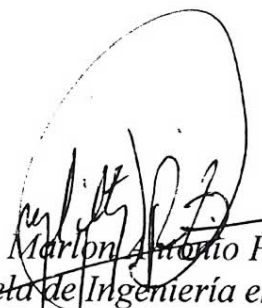
C  
I  
E  
N  
C  
I  
A  
S


Y

S  
I  
S  
T  
E  
M  
A  
S

*El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación “DEFINICIÓN DE LA ARQUITECTURA SKRN”, realizado por el estudiante OSCAR ESTUARDO DE LA MORA HERNÁNDEZ, aprueba el presente trabajo y solicita la autorización del mismo.*

“ID Y ENSEÑAD A TODOS”

  
Ing. Marlon Antonio Pérez Turk  
Director, Escuela de Ingeniería en Ciencias y Sistemas



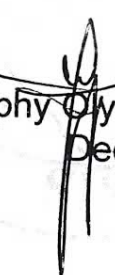
Guatemala, 01 de Octubre 2013



Ref.DTG.676.2013

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ciencias y Sistemas, al trabajo de graduación titulado: **DEFINICIÓN DE LA ARQUITECTURA SKRN**, presentado por el estudiante universitario: **Oscar Estuardo de la Mora Hernández**, procede a la autorización para la impresión del mismo.

IMPRÍMASE.

  
Ing. Murphy Olympo Paiz Recinos  
Decano



Guatemala, octubre de 2013

/cc



## **ACTO QUE DEDICO A:**

- Mis padres** Margarita Amelia Hernández y Oscar Ramón de la Mora, por apoyarme en todo lo que pudieron.
- Mis hermanos** Massiel y Alejandro de la Mora, por echarme ánimos y ganas en mis desvelos.
- Mis tíos y primos** Tío Juan, tía Tere, Andrea y Juan Rosales, tío Enrique, tía Isabel, Celia, Katty, Anabela, Ligia y Enrique Gordillo, por ser parte de mi formación, de lo que soy ahora.
- Mi tío** Oscar de la Mora, que en mis situaciones más difíciles, estuvo apoyándome incondicionalmente.
- Mis abuelas** Susana Reyes, Amparo Rodas y Blanca Cornejo (Q.e.p.d), que hubiera sido de mí sin sus cuidados y sus mesadas extras, las quiero mucho.

## **AGRADECIMIENTOS A:**

<b>La Universidad de San Carlos de Guatemala</b>	Por ser el lugar donde obtuve mi formación, no solo académica, también personal y social.
<b>Facultad de Ingeniería</b>	Porque me permitió desarrollar mis habilidades.
<b>Mis primeros amigos de la Universidad</b>	Renato Pereira, Julio Vargas, Gustavo Ayapán, Amel Ruiz, por sus ánimos, consejos y ayuda.
<b>La Organización</b>	Alicia Austin, Anastasia del Cid, Deneve Alejos, Helen Luarca y Galatea Rodríguez, por su cariño, apoyo y por saber que siempre que puedan, la mayoría, estarán ahí.
<b>Mis amigos del trabajo</b>	Edsson Gonzalez, Mario Velásquez, Carlos Nuñez, Zoila Castellanos y Lourdes Membreño, por ayudarme en mi experiencia profesional.
<b>Mis amigos de la infancia</b>	Francisco Antonio, Lobsiguer Perez, Eddy Girón, Enrique Salas; tantas cosas que compartimos y hablamos y ahora las realizamos.

## ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES.....	VII
GLOSARIO.....	XI
RESUMEN.....	XIX
OBJETIVOS .....	XXI
INTRODUCCIÓN.....	XXIII
1. MARCO TEÓRICO.....	1
1.1. Software .....	1
1.2. Ingeniería de software .....	2
1.3. Arquitectura de software .....	3
1.4. Componentes de software .....	3
1.5. Servicios web .....	4
1.5.1. SOA.....	4
1.5.2. WSDL .....	4
1.6. Patrones de diseño.....	5
1.6.1. Factory Method.....	5
1.6.2. Facade.....	6
1.6.3. Strategy .....	6
1.6.4. Adapter .....	6
1.7. Bases de datos.....	7
1.7.1. Entidades.....	7
1.7.2. Atributos.....	8
1.7.3. Relaciones .....	9
1.8. Patrón MVC.....	9
1.8.1. Capa modelo .....	10

1.8.2.	Capa controlador .....	10
1.8.3.	Capa vista .....	10
1.8.3.1.	Javascript .....	10
1.8.3.2.	Cascading Style Sheet.....	11
1.8.3.3.	HTML.....	11
2.	VISIÓN GENERAL .....	13
2.1.	Capa de modelo .....	14
2.2.	Capa de controlador.....	15
2.2.1.	Capa de acceso a datos .....	15
2.2.2.	Capa de control .....	15
2.3.	Capa de vista .....	16
2.3.1.	Capa de presentación .....	16
2.3.2.	Capa de interfaz .....	16
2.4.	Sistema de usuarios y permisos.....	16
3.	CAPA DE ACCESO A DATOS .....	19
3.1.	Objetos transaccionales: MyBDItem.....	21
3.1.1.	Validación y sanación de datos .....	25
3.1.2.	Mensajes de error y subclases .....	25
3.2.	Objetos de consultas: MyBDQuery.....	26
3.2.1.	Protección contra inyección SQL.....	28
3.3.	Bitácora de datos .....	29
3.3.1.	Estructura .....	29
4.	CAPA DE CONTROL.....	31
4.1.	Interfaz de exposición de clases y métodos: WSLib .....	34
4.1.1.	Conexión por instancia .....	34
4.1.2.	Conexión por servicio web.....	36

4.2.	Librerías auxiliares.....	37
4.2.1.	SKRN-API.....	38
4.2.2.	Generador de gráficas: jPGraph .....	38
5.	CAPA DE VISTA.....	39
5.1.	Organización de la presentación.....	40
5.2.	Javascript y archivos de interfaz .....	42
5.3.	Sesiones.....	44
5.3.1.	Permisos de visualización .....	45
5.3.2.	Permisos de operación .....	46
5.4.	Ayuda en línea.....	47
5.5.	Librerías Javascript.....	47
6.	SISTEMA DE USUARIOS Y PERMISOS .....	49
6.1.	Estructuración de sistemas .....	49
6.1.1.	Módulo .....	49
6.1.2.	Opción .....	49
6.1.3.	Permisos.....	50
6.2.	Asignación de permisos a roles .....	51
6.3.	Usuarios .....	51
6.3.1.	Autenticación .....	52
6.3.2.	Verificación de permisos de visualización .....	53
6.3.3.	Verificación de permisos de operación.....	54
6.3.4.	Obtención de menú.....	55
6.3.5.	Carga de datos de sesión .....	55
6.3.6.	Sistemas en otros paradigmas .....	56
6.3.7.	Sistemas en servidores remotos .....	56
6.3.8.	Aplicaciones de escritorio .....	57
6.4.	Documentación de sistemas .....	58

6.4.1.	Widget de ayuda.....	58
6.4.2.	Generación de manuales de usuario .....	58
6.4.3.	Generación de manuales técnicos.....	59
6.5.	Despliegue de capas de arquitectura .....	59
6.5.1.	Un servidor .....	59
6.5.2.	Dos servidores .....	60
6.5.3.	Tres o más servidores .....	62
6.6.	Interacción entre capas .....	64
7.	IMPLEMENTACIÓN EN LA INSTITUCIÓN DEL REGISTRO GENERAL DE LA PROPIEDAD .....	67
7.1.	Registro General de la Propiedad .....	67
7.2.	Implementación.....	68
7.2.1.	Integración con aplicaciones anteriores y de terceros .....	68
7.2.2.	Seguridad.....	70
7.3.	Métricas de mejora de proceso .....	71
7.3.1.	Despliegue de la arquitectura .....	73
7.4.	Experiencia de implementación.....	74
8.	EVALUACIÓN DE ARQUITECTURA.....	77
8.1.	Presentación .....	77
8.1.1.	Presentación de ATAM.....	77
8.1.2.	Presentación de objetivos del negocio.....	79
8.1.2.1.	Facilitar la capacitación, documentación del uso de los sistemas .....	80
8.1.2.2.	Descentralizar de servicios de la institución.....	80

8.1.2.3.	Controlar la corrupción y seguridad de la información .....	81
8.1.2.4.	Mejorar del “Doing Bussiness” de la institución .....	82
8.1.2.5.	Modernizar la tecnología de la institución .....	83
8.1.3.	Presentación de arquitectura .....	83
8.2.	Investigación y análisis .....	83
8.2.1.	Identificación de propuestas arquitectónicas .....	84
8.2.2.	Generación de árbol de atributos de utilidad .....	84
8.2.3.	Análisis de propuestas arquitectónicas .....	92
8.3.	Testeo .....	108
8.3.1.	Lluvia de ideas y priorización de escenarios .....	108
8.3.2.	Análisis de propuestas arquitectónicas .....	109
8.4.	Reporte.....	115
8.4.1.	Resultados de análisis .....	115
CONCLUSIONES .....		121
RECOMENDACIONES.....		123
BIBLIOGRAFÍA.....		125





## ÍNDICE DE ILUSTRACIONES

### FIGURAS

1.	Vista de las capas y subcapas de la arquitectura .....	13
2.	Vista de despliegue de la arquitectura.....	14
3.	Vista lógica del acceso a datos .....	19
4.	Vista de despliegue del acceso a datos .....	20
5.	Vista física del acceso a datos y los datos .....	20
6.	Vista lógica del uso del objeto transaccional .....	22
7.	Vista de procesos de objetos transaccionales .....	24
8.	Vista lógica del uso de los objetos de consulta .....	27
9.	Vista de procesos de objetos de consulta .....	28
10.	Esquema E-R de la bitácora.....	30
11.	Vista lógica del diagrama de clases .....	32
12.	Vista de despliegue de la capa de control .....	33
13.	Vista física de la conexión por instancia.....	34
14.	Vista de procesos de la conexión por instancia .....	35
15.	Vista física de la conexión por servicio web .....	36
16.	Vista de procesos de conexión por servicio web .....	37
17.	Vista de despliegue.....	39
18.	Vista de procesos de una página .....	40
19.	Vista lógica de la capa de vista .....	41
20.	Vista de procesos de enviar y recibir información.....	43
21.	Envío y recuperación de información con WS .....	44
22.	Vista de procesos. Permiso de visualización .....	45
23.	Vista de procesos. Permiso de operación .....	46

24.	Vista gráfica de un sistema.....	50
25.	Vista de proceso de autenticación .....	52
26.	Vista de proceso de permisos de visualización.....	53
27.	Vista de procesos de permisos de operación.....	54
28.	Vista de procesos de obtención de menú .....	55
29.	Vista de despliegue del sistema de usuarios y permisos .....	56
30.	Vista física .....	57
31.	Diagrama de despliegue con un servidor.....	60
32.	Diagrama de despliegue con dos servidores .....	61
33.	Diagrama de despliegue 3 servidores.....	62
34.	Diagrama de despliegue 3 o más servidores .....	63
35.	Diagrama de despliegue de aplicaciones distribuidas.....	64
36.	Proceso de comunicación entre capas .....	65
37.	Diagrama de aplicación de escritorio .....	70
38.	Diagrama de despliegue de servidores.....	73
39.	Diagrama de pasos del proceso de análisis ATAM.....	78
40.	Objetivos del negocio .....	79
41.	Diagrama de árbol de utilidad .....	86

## TABLAS

I.	Parámetros del métodos de operación .....	23
II.	Contabilización de aplicaciones y módulos del RGP.....	68
III.	Atributos de calidad requeridos en la institución .....	87
IV.	Tabla de escenarios de mayor relevancia.....	90
V.	Centralización de usuarios, sistemas, roles y permisos .....	92
VI.	Uso del patrón MVC .....	94
VII.	Implementación de bitácoras de manipulación de datos.....	95
VIII.	Implementación de bitácoras de ingresos.....	97

IX.	Servicio web de autenticación de ingresos y permisos.....	98
X.	Uso de patrón vista modelo controlador .....	100
XI.	Uso del patrón modelo vista controlador .....	102
XII.	Uso de acceso a datos para consulta y transacciones .....	103
XIII.	Comunicación de la vista con el controlador por servicio web .....	104
XIV.	Centralización de estilos CSS para las vistas de la aplicación .....	106
XV.	Tabla de escenarios no considerados .....	108
XVI.	Tabla de escenarios de mayor relevancia .....	109
XVII.	Servicios web de comunicación entre sistemas internos y externos...	110
XVIII.	Versiones de aplicación .....	111
XIX.	Replicación de datos .....	113



## GLOSARIO

<b>Acoplamiento</b>	Nivel de dependencia entre las unidades de software de un sistema informático.
<b>Algoritmo</b>	Conjunto de instrucciones bien definidas para realizar una actividad.
<b>API</b>	Application Programming Interface.
<b>Aplicación</b>	Conjunto de algoritmos integrados para realizar una tarea específica.
<b>Arreglo</b>	Conjunto de datos homogéneos que se encuentran ubicados de forma consecutiva en la memoria volátil.
<b>Bitácora</b>	Conjunto de datos históricos de consulta ordenados cronológicamente.
<b>Capas</b>	Estructura lógica de un software que realiza tareas muy específicas en una aplicación.
<b>Clase</b>	Archivo que describe la estructura de un objeto definiéndose en ella, sus atributos y métodos.

<b>Código fuente</b>	Conjunto de líneas de texto que son las instrucciones que debe seguir un programa para realizar alguna tarea.
<b>Cohesión</b>	Forma en que se agrupan las unidades de software en una unidad mayor.
<b>Compatibilidad</b>	Condición o habilidad que tiene un programa o un sistema para adherirse correctamente a otro software.
<b>Configuración</b>	Conjunto de datos que determina el valor de algunas variables de un programa o sistema.
<b>Constructor</b>	Método de una clase que inicializa los valores iniciales de un objeto.
<b>Consulta</b>	Instrucción en lenguaje SQL para la realización de consultas de datos en una base de datos.
<b>Credencial</b>	Conjunto de datos que autentican o verifican la identidad de un usuario.
<b>CSS</b>	Cascade Style Sheet.
<b>DBMS</b>	DataBase Management System.
<b>Descodificación</b>	Operación inversa a la de la codificación, esta transforma datos a su forma original.

<b>Documentación</b>	Conjunto de información escrita o electrónica que contiene los pasos a seguir de la utilización de un producto.
<b>E-R</b>	Entidad Relación.
<b>Encriptación</b>	Algoritmo de seguridad que permite transformar datos en información difícil de entender para su transmisión en conexiones no seguras.
<b>Entidad</b>	Es la representación de un objeto o de un concepto del mundo real que se describe en una base de datos.
<b>Error</b>	Es un fallo en un programa de un sistema de software, que desencadena un resultado indeseado.
<b>Excepción</b>	Es un fallo en un programa que desencadena un resultado indeseado, pero a diferencia de un error, puede ser manejado ejecutándose un bloque de instrucciones que sustituirá al resultado del error.
<b>Fila</b>	Es un conjunto de campos en la tabla de una base de datos.
<b>Hardware</b>	Es la parte física de un sistema de cómputo.
<b>Herencia</b>	Característica esencial de los objetos que permite la reutilización de código.

<b>HTML</b>	Hyper Text Markup Language.
<b>ID</b>	Identificador único.
<b>Instancia</b>	Se refiere a la realización específica de una clase o prototipo determinado.
<b>Institución pública</b>	Es aquella que es administrada, financiada y controlada por el Gobierno de un país.
<b>Integridad</b>	Es la corrección o complementación de los datos de una base de datos.
<b>Interfaz</b>	Componente de software utilizado para la comunicación entre componentes y sistemas informáticos.
<b>JS</b>	Javascript.
<b>Librería</b>	Conjunto de clases que tienen alta cohesión en su funcionamiento.
<b>Metadato</b>	Son datos sobre datos, datos estructurados que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas.
<b>MVC</b>	Modelo Vista Controlador.



<b>Objeto</b>	Es una unidad dentro de un programa de computadora, que consta de un estado y un comportamiento.
<b>Parámetro</b>	Es un tipo de variable que es recibida por una función, procedimiento o subrutina.
<b>Permiso</b>	Es la autorización que se le otorga a un usuario o programa para realizar una operación sobre los datos.
<b>Registro</b>	Representa un objeto único de datos en una base de datos.
<b>Requerimiento</b>	Atributo necesario dentro de un sistema que representa una capacidad, característica o un factor de calidad.
<b>RGP</b>	Registro General de la Propiedad.
<b>Rol</b>	Es una agrupación de permisos asociados a un usuario o sistema para la realización de consultas y manipulación de datos.
<b>Sanación de datos</b>	Es la verificación y corrección de datos que se vayan a ingresar a una base de datos.

<b>Servicio</b>	Es una actividad o consecuencia de un sistema informático para proveer utilidades o beneficios a otros sistemas o usuarios.
<b>Servidor</b>	Es un nodo que pertenece a una red y provee de servicios a otros nodos denominados clientes.
<b>Sesión</b>	Es la duración de una conexión a un determinado sistema o red que suele incluir el intercambio de paquetes de información entre un usuario y un servidor.
<b>Sistema remoto</b>	Sistema informático que no se encuentra en la misma área física de red de una empresa o institución.
<b>Software</b>	Es el equipamiento lógico de un sistema informático.
<b>SQL</b>	Structured Query Language.
<b>Superclase</b>	Es una clase que hereda atributos y métodos a otras clases que heredan de ella.
<b>Transacción</b>	Es un conjunto de órdenes que se ejecutan formando una unidad de trabajo, es decir, en forma indivisible o atómica.
<b>UML</b>	Unified Modeling Language.

<b>URL</b>	Uniform Resource Locator.
<b>Versionamiento</b>	Es la gestión de diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo.
<b>WS</b>	Web Service.
<b>XML</b>	eXtensible Markup Language.



## RESUMEN

Un sistema informático es aquel que basa su procesamiento en el empleo de la computación, con funciones interrelacionadas como hardware, software y recurso humano. Estos pasan por diferentes fases en su ciclo de vida, desde la captura de requerimientos, hasta el mantenimiento. Los sistemas pueden ser desarrollados mediante muchos métodos y paradigmas de programación.

La arquitectura de software es una representación de alto nivel de la estructura de un sistema, describe sus partes, las interacciones, composición y restricciones. Establece principios de ingeniería del software robustos y así obtener un software económico y fiable.

Una arquitectura está basada en patrones de diseño, este en particular, se basa en el patrón MVC, proveyendo de un acceso a datos, mecanismo de control y programación de algoritmos, un mecanismo de despliegue de vistas y un sistema que centraliza la información de usuarios, sistemas, roles, permisos y documentación técnica y de usuario.

Se realizará una evaluación de arquitectura y así mostrar sus atributos de calidad.



## **OBJETIVOS**

### **General**

Explicar, presentar y evaluar una arquitectura de software basada en el patrón MVC, desarrollada en PHP para el desarrollo de software.

### **Específicos**

1. Presentar cómo se divide la arquitectura en el patrón MVC.
2. Presentar cada subcapa de las capas MVC de la arquitectura a presentar.
3. Explicar cómo los patrones de diseño se implementaron para un desarrollo eficaz.
4. Explicar el uso de servicios web para la comunicación entre los sistemas.
5. Explicar el por qué de la división de las capas en la arquitectura en subcapas con tareas específicas.
6. Explicar el uso de pruebas unitarias para la evaluación del funcionamiento de la aplicación.
7. Evaluar los componentes de software para ser reutilizados en los distintos sistemas de la institución independiente de su contexto.

8. Evaluar los componentes de acceso a datos para la manipulación de estos mismos.
9. Explicar el por qué de usar un sistema de permisos, usuarios, documentación de forma centralizada.
10. Evaluar la sensibilidad de la arquitectura.



## INTRODUCCIÓN

Este trabajo se enfocará en la descripción, evaluación y explicación de una arquitectura de software, desarrollada en lenguaje PHP, donde se describirán las capas y subcapas, la comunicación entre estas, el funcionamiento del acceso a datos, la seguridad, integridad, disponibilidad de estos y la importancia de usar un sistema que integre la información de los sistemas, sus componentes, usuarios y roles de acceso.

Se representará la estructura de cada componente y capa por medio de lenguaje UML, utilizando distintas vistas, teniendo así, una visión amplia del funcionamiento de la arquitectura.

Esta arquitectura permite un desarrollo modular, una baja curva de aprendizaje y fácil acoplamiento con aplicaciones anteriormente creadas, incluso con distintos paradigmas de programación, haciendo de esta, una herramienta versátil, escalable, modificable, segura e integrable.



# 1. MARCO TEÓRICO

## 1.1. Software

Refiriéndose a una definición más amplia, el software, de acuerdo con Ian Sommerville, en su libro Ingeniería del Software, es, además de los programas de computadora, los documentos asociados y la configuración de datos que se necesitan para hacer que estos operen de manera correcta, por lo general un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema y sitios web que permitan al usuario descargar información de productos recientes. Existen dos tipos de productos de software:

- Productos genéricos: sistemas aislados producidos por una organización de desarrollo y venden al mercado abierto a cualquier cliente que pueda comprarlos.
- Productos personalizados: sistemas requeridos por un cliente en particular. Un contratista de software desarrolla el software especialmente para ese cliente.

Una diferencia importante entre estos tipos, es que los productos genéricos, la organización desarrolladora controla su especificación, y los productos personalizados, por lo general es desarrollada y controlada por la organización que compra el software.

## **1.2. Ingeniería de software**

Es una disciplina de la ingeniería, según Ian Sommerville, que comprende todos los aspectos de la producción de software, desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de este después de que se utiliza. Existen dos frases clave:

Disciplina de la ingeniería: los ingenieros hacen que las cosas funcionen. Aplican teorías, métodos y herramientas donde sean convenientes, pero las utilizan de forma selectiva, y siempre tratando de descubrir soluciones a los problemas, aun cuando no existan teorías y métodos aplicables para resolverlos. Los ingenieros también saben que deben trabajar con restricciones financieras y organizacionales, por lo que buscan soluciones tomando en cuenta estas restricciones.

Todos los aspectos de producción de software. La ingeniería del software no solo debe comprender los procesos técnicos del desarrollo, si no también, actividades como gestión de proyectos de software y desarrollo de herramientas, métodos y teorías de apoyo a la producción de software.

En general, los ingenieros de software adoptan un enfoque sistemático y organizado en su trabajo, ya que es la forma más efectiva de producir software de alta calidad. Sin embargo, aunque la ingeniería consiste en seleccionar el método más apropiado para un conjunto de circunstancias, un enfoque más informal y creativo de desarrollo podría ser efectivo en algunas circunstancias.

### **1.3. Arquitectura de software**

La arquitectura de software, conforme con Isidro Ramos Salavert, en su libro Ingeniería del software y bases de datos: tendencias actuales, es la representación de alto nivel de la estructura de un sistema o aplicación, que describe sus partes que la integran, las interacciones entre ellas, los patrones que supervisan su composición y las restricciones a la hora de aplicar dichos patrones. Establece y usa principios de ingeniería robustos, orientados a obtener un software económico que sea fiable y funcione de manera eficiente sobre máquinas reales, según Roberto Cortez Morales. Lo componen mecanismos y herramientas que permiten la creación e interconexión de componentes de software, junto a una colección de servicios para facilitar las labores de dichos componentes.

### **1.4. Componentes de software**

Un componente, de acuerdo con Isidro Ramos Salavert, es una unidad de composición de aplicaciones de software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente en tiempo y espacio. Es importante señalar que un componente se puede reutilizar, no significa usarlo más de una vez, sino que implica la capacidad de ser utilizado en contextos distintos a aquellos para los que fue diseñado.

## **1.5. Servicios web**

Es un conjunto, conforme con Santiago Marqu ez Sol s, en su libro La web sem ntica, de aplicaciones o tecnolog as con capacidad de interactuar en la web. Estas aplicaciones o tecnolog as intercambian datos entre s  con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos.

Estos servicios proporcionan mecanismos de comunicaci n est ndares entre diferentes aplicaciones, que interact an entre s  para presentar informaci n din mica al usuario.

### **1.5.1. SOA**

Es una arquitectura de referencia est ndar, caracterizada por la utilizaci n de servicios para dar soporte a requerimientos de software de los usuarios, para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones y que al mismo tiempo, sea posible su combinaci n y realizar operaciones complejas.

SOA se identifica por la utilizaci n de protocolos como SOAP y WSDL en su implementaci n, estos servicios son d bilmente acoplados y altamente interoperables.

### **1.5.2. WSDL**

Definici n formal y est ndar, independiente de la plataforma subyacente y del lenguaje de programaci n o la tecnolog a de desarrollo, encapsula las

particularidades de una implementación, un servicio web programado en lenguaje C# por ejemplo, puede ser usado por una aplicación Java.

## **1.6. Patrones de diseño**

Son esqueletos de las soluciones, de acuerdo con Elizabeth Freeman, Eric Freeman y Bert Bates, en su libro *Head first design patterns*, a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada a problemas que están sujetos a contextos similares. Se debe tener presente los siguientes elementos en un patrón:

- Nombre
- El problema
- La descripción abstracta del problema
- Las consecuencias

Existen varios tipos de patrones de diseño y se clasifican en:

- Creacionales: inicialización y configuración de objetos.
- Estructurales: separan la interfaz de la implementación. Se ocupan de como las clases y objetos se agrupan para formar estructuras más grandes.
- Comportamiento: más que describir objetos o clases, describen la comunicación entre ellos.

### **1.6.1. Factory Method**

Centralizado en una clase constructora para la creación de objetos de un subtipo de un tipo determinado, ocultando al cliente la casuística para elegir el

subtipo a crear, así, no se provee la clase de objetos que debe crear, se centraliza y provee una interfaz para el cliente, permitiendo crear una familia de objetos sin especificar su clase.

### **1.6.2. Facade**

Provee una interfaz simple para un subsistema complejo, o cuando se quiere estructurar varios subsistemas en capas, ya que las fachadas serian el punto de entrada a cada nivel.

### **1.6.3. Strategy**

Se utiliza para manejar la selección de un algoritmo en tiempo de ejecución. Se compone de:

- Contexto: elemento que usa los algoritmos, por lo tanto, delega la jerarquía de estrategias.
- Estrategia: interfaz común para todos los algoritmos soportados. Esta será usada por el contexto para invocar a la estrategia concreta.
- Estrategia concreta: implementa el algoritmo usando la interfaz definida por la estrategia.

### **1.6.4. Adapter**

Convierte la interface de una clase en otra interface que el cliente espera. El adaptador funciona junto a las demás pues estas dos otras son incompatibles.



## **1.7. Bases de datos**

Es un conjunto de datos, según Lucía I. Cardoso M., en su libro Sistemas de bases de datos II, relacionados entre sí, modelan o reflejan información en una organización. Esta información es manipulada por los sistemas de gestión de bases de datos, conforme con Olga Pons Capote, en su libro Introducción a los sistemas de bases de datos, este tiene operaciones básicas que son:

- Recuperación de registros por clave de búsqueda
- Obtención del siguiente registro
- Inserción de registros
- Actualización de registros
- Lectura de todo el archivo
- Reorganización del archivo

El modelo conceptual representa una visión global de una base de datos y es el principio para la identificación y la descripción de los objetos de datos principales, sin entrar en detalles. El modelo conceptual más utilizado es el Entidad-Relación y sus componentes son:

### **1.7.1. Entidades**

En el modelo E-R se refiere a una fila de tabla específica como una instancia de entidad. El nombre de la entidad, un sustantivo, casi siempre se escribe con mayúscula y en singular.

### 1.7.2. Atributos

Son las características de las entidades y tienen las siguientes características:

- **Dominios:** es el conjunto de posibles valores del atributo, por ejemplo, el dominio del atributo numérico calificación promedio, se escribe (0, 4) pues el valor más bajo posible es 0 o 4. Los atributos pueden compartir un dominio, por ejemplo, el dominio de un estudiante y el de un profesor comparten el mismo dominio de posibles domicilios uso, se utiliza el mismo nombre del atributo, por ejemplo, las entidades profesor y estudiante pueden tener un atributo llamado domicilio.
- **Claves primaria:** se utilizan para identificar de forma única las filas de una tabla. Idealmente se compone de un solo atributo, pero puede ser posible usar una clave compuesta, es decir, una clave primaria formada por más de un atributo.
- **Atributos compuestos y simples:** un atributo compuesto no debe confundirse con clave compuesta, es el que se puede subdividir en más atributos adicionales, por ejemplo, la dirección que puede dividirse en calle, ciudad, estado, etc. Un atributo simple no se puede dividir y generalmente es apropiado usar los atributos compuestos en una serie de atributos simples.
- **Atributos de un solo valor:** es el que puede tener solamente un valor, por ejemplo, una persona solo puede tener un número de seguro social y una pieza manufacturada solo puede tener un número de serie.

- Atributos de valores múltiples: son los que pueden tener muchos valores, por ejemplo, una persona puede tener muchos grados académicos, o puede tener varios números de teléfono diferentes, cada uno con su propio número.

### **1.7.3. Relaciones**

Definen la forma más idónea de enlazar la información de las diferentes entidades. Es una asociación entre entidades, las entidades que participan en una relación se conocen como participantes. Las relaciones se clasifican según su cardinalidad:

- Uno a uno: una entidad puede estar relacionada solamente con otra entidad y viceversa.
- Uno a muchos: es la relación ideal y constituye el principal bloque de construcción de la base de datos relacional. Una entidad puede estar relacionada con muchos valores de otra entidad.
- Muchos a muchos: es la más problemática en el ambiente relacional, pero se puede ejecutar si se descompone para producir un conjunto de relaciones uno a muchos. Se crea una entidad llamada compuesta o entidad puente, vincula las tablas de la relación muchos a muchos y en su estructura incluye por lo menos las claves primarias de las tablas que se han de vincular.

### **1.8. Patrón MVC**

Es un patrón de diseño estructural, donde las funcionalidades de la aplicación se separan en capas con tareas específicas.

### **1.8.1. Capa modelo**

Describe, de acuerdo con Aroa Solana, en su libro *Windows Communication Foundation*, las entidades propias del dominio del problema que resuelve la aplicación, en esta capa aparecen los llamados objetos de negocio que abstraen las características del dominio del problema.

### **1.8.2. Capa controlador**

Crea un puente entre la capa de modelo y vista, es decir, por parte del modelo es posible tener un conjunto de clases para modelar la información del dominio del problema. Por otra parte elementos que tienen que ver con lo visual, como controles, figuras, imágenes y efectos, pueden usarse en aplicaciones que no necesariamente estén ligadas a un modelo de datos.

### **1.8.3. Capa vista**

Describe la apariencia y funcionalidad de la interfaz de usuario. Captura desde la interfaz de usuario y recibe información desde la capa de controlador para ser mostrada a este. La capa de vista se divide en distintos elementos:

#### **1.8.3.1. Javascript**

Es un lenguaje orientado a objetos ligero, más conocido como el lenguaje de script para páginas web, pero también usado en muchos entornos sin navegador. Este tiene tipos y operadores, objetos principales y métodos.

Su sintaxis, de acuerdo con Wilison, en su artículo Una reintroducción a Javascript, se basa de lenguajes como Java, C y muchas estructuras provienen

de estos lenguajes. Una de las diferencias principales es que no existen clases, en cambio, la clase se maneja como prototipos de objetos. Otra diferencia es que las funciones son objetos, proveyendo la capacidad de mantener el código ejecutable como si fuera otro objeto.

### **1.8.3.2. Cascading Style Sheet**

Es un mecanismo, simple para agregar estilos, colores, fuente, espacios a documentos web.

Abarca cuestiones relativas a fuentes, colores, márgenes, líneas, altura, anchura, imágenes de fondo, posicionamiento y muchos otros temas. Añadir formato a los sitios web es preciso y sofisticado y CSS lo provee. Mientras HTML estructura la web, CSS la formatea y ofrece:

- Control de la presentación de muchos documentos desde una única hoja de estilo.
- Control más preciso de la presentación.
- Aplicación de diferentes presentaciones a diferentes tipos de medios (pantalla, impresión, etc).
- Numerosas técnicas avanzadas y sofisticadas.

### **1.8.3.3. HTML**

Hyper Text Markup Language es una publicación del lenguaje de la internet. Es uno de los principales componentes de la plataforma web. La primera versión fue descrita por Tim Berners-Lee al final de 1991. La versión recomendada hasta el momento es HTML 4.01, publicada en diciembre del 1999. Y ahora se trabaja en la siguiente versión HTML5.

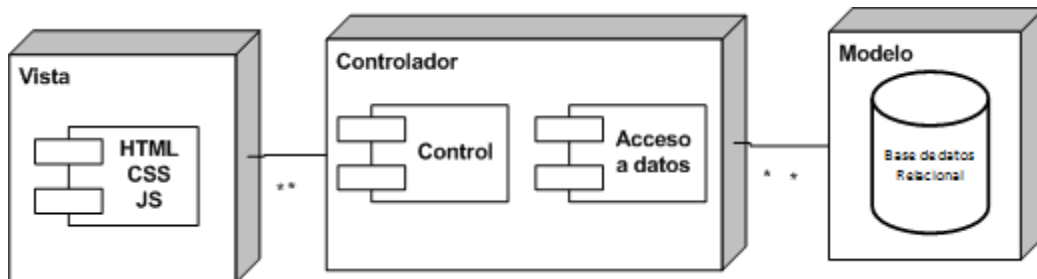
Hace referencia al lenguaje de marcado predominante en la elaboración de páginas web y se usa para describir y traducir la estructura y la información. El HTML se escribe en forma de etiquetas, también puede describir hasta un cierto punto, la apariencia de un documento, normalmente se complementa con un script como javascript para el comportamiento del sitio y css para su apariencia y estilo.

## 2. VISIÓN GENERAL

Este capítulo presenta de forma general la arquitectura SKRN, describiendo su estructura y componentes.

La arquitectura SKRN está basada en el patrón MVC, compuesta por tres capas. Separa las funciones del código especializando cada una de estas con tareas específicas, que podrían ser tareas transaccionales, ejecución de algoritmos, seguridad, comunicación, entre otras.

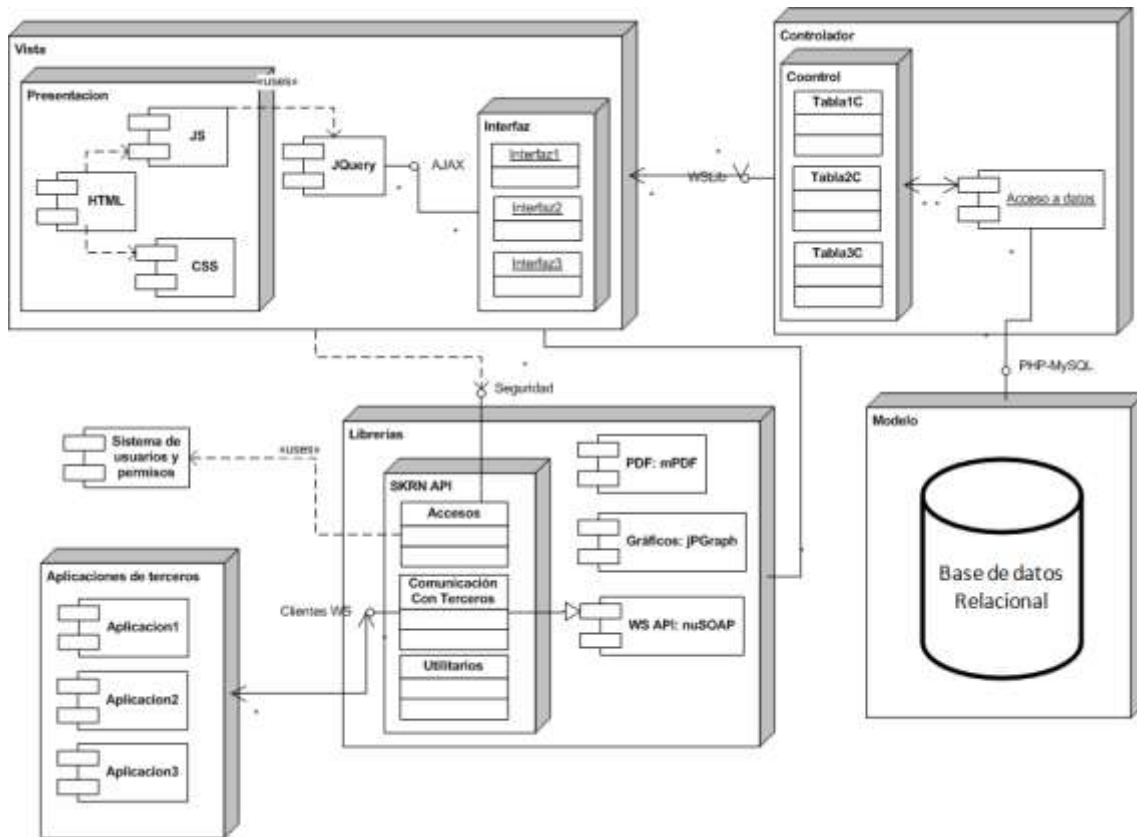
Figura 1. **Vista de las capas y subcapas de la arquitectura**



Fuente: elaboración propia con MS Visio 2007.

El siguiente diagrama muestra la organización de las capas y componentes de la arquitectura en cuestión.

Figura 2. Vista de despliegue de la arquitectura



Fuente: elaboración propia con MS Visio 2007.

## 2.1. Capa de modelo

Proporcionado por el DBMS, ya que es una arquitectura escrita en PHP, utiliza preferiblemente MySQL con el motor de almacenamiento InnoDB para el manejo de información. Se conecta hacia la capa de controlador por medio TCP/IP.



## **2.2. Capa de controlador**

Esta capa se encarga de la lógica del negocio, los algoritmos y acceso a la información de la base de datos, está dividida en dos subcapas:

### **2.2.1. Capa de acceso a datos**

Es una interfaz que facilita el acceso y manipulación a la información de la base de datos, proporciona capacidad transaccional, simplifica la extracción de los datos de las consultas y provee una bitácora para mantener un historial de la manipulación de los datos, utiliza la extensión mysqli, y si esta no está disponible, probablemente por una usar una versión antigua de PHP, utiliza la antigua extensión mysql.

Esta subcapa no se comunica directamente con la capa de vista, lo hace a través de la subcapa de control mediante instancias de los objetos de esta. La capa de acceso a datos se divide en dos partes:

- Transacciones y mantenimiento de datos
- Consultas a la base de datos

### **2.2.2. Capa de control**

Encargado de almacenar y ordenar los algoritmos, la lógica del negocio, establecer métodos predeterminados para la manipulación de información, ejecuta los objetos transaccionales, invoca las consultas y construye mensajes XML de datos que se envían a la capa de vista. La comunicación con la vista depende de la configuración de la arquitectura, se utilizan llamadas mediante instancias o mediante servicio web.

## **2.3. Capa de vista**

Se encarga del manejo de sesiones, despliegue y envío de información. Cabe mencionar que la arquitectura puede publicarse de dos modos, la capa de vista y de controlador pueden estar en el mismo servidor, o bien puede estar en servidores diferentes (estas configuraciones se mostrarán más adelante en el capítulo 5), si este es el caso, existe un componente de servicio web para la comunicación entre las dos capas. La capa de vista está dividida en dos subcapas:

### **2.3.1. Capa de presentación**

Encargada de desplegar información, recolectar y cargar datos de formularios, generación de reportes y manejo de sesiones para permisos de visualización e ingreso. Está compuesta mayormente por HTML, css y javascript, se auxilia con librerías en javascript que interpretan mensajes XML enviados desde el controlador y despliegan la información contenida en el.

### **2.3.2. Capa de interfaz**

Encargada de conectar la vista con el controlador al invocar los métodos de este, recibe y envía mensajes de la capa de presentación, mayormente a través de AJAX y controla el manejo de sesiones para permisos de operación.

## **2.4. Sistema de usuarios y permisos**

Sistema encargado de gestionar la estructura de los sistemas, roles, asociación de permisos, centralización de usuarios y generación de documentación. Este sistema está escrito sobre la arquitectura a describir e

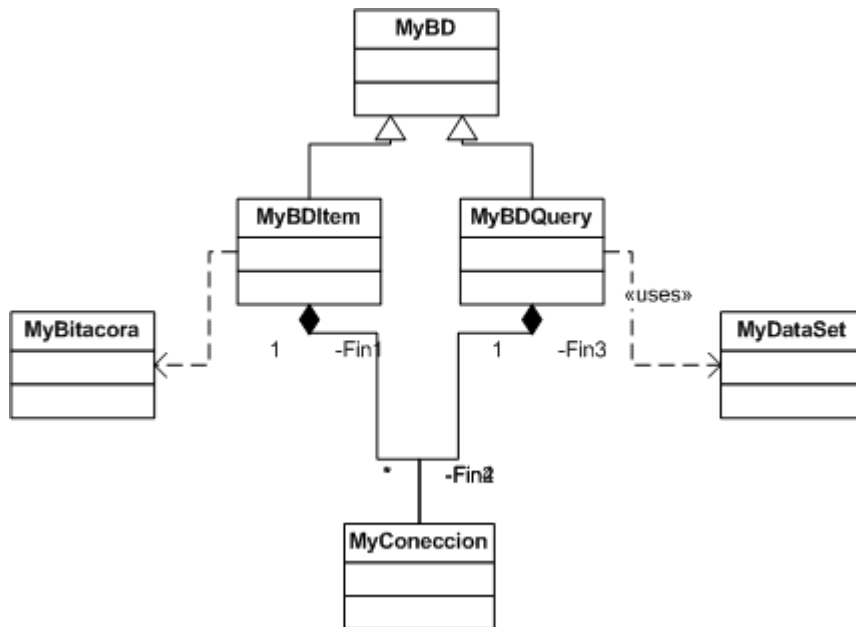
implementa todas las características a mencionar. Permite crear accesos y permisos a sistemas informáticos tanto locales como remotos y/o de terceros, estos pueden implementar la arquitectura o bien que tengan la capacidad de adaptar la estructura de los permisos de los sistemas.



### 3. CAPA DE ACCESO A DATOS

Se encarga del acceso a los datos en las aplicaciones, provee transacciones, consultas, mantenimiento de tablas, gestión de conexiones a la base de datos, protección, validación e integridad de datos. El siguiente diagrama muestra los componentes de esta capa.

Figura 3. Vista lógica del acceso a datos

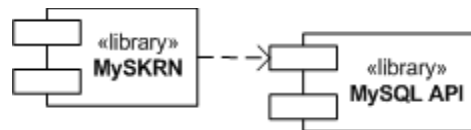


Fuente: elaboración propia con MS Visio 2007.

Provee de clases para facilitar el acceso a las tablas, contiene sus propias excepciones para el manejo de errores y provee de una bitácora para una vista detallada del control de los cambios en los datos de la base de datos.

MySKRN es un componente que utiliza el API de acceso de base de datos de MySQL en el lenguaje de PHP, simplifica el modo de conexión separando esta de las transacciones con las consultas, la idea principal es que las consultas se agrupen según la tabla principal que se vaya a consultar. El siguiente diagrama muestra la relación de las librerías mencionadas.

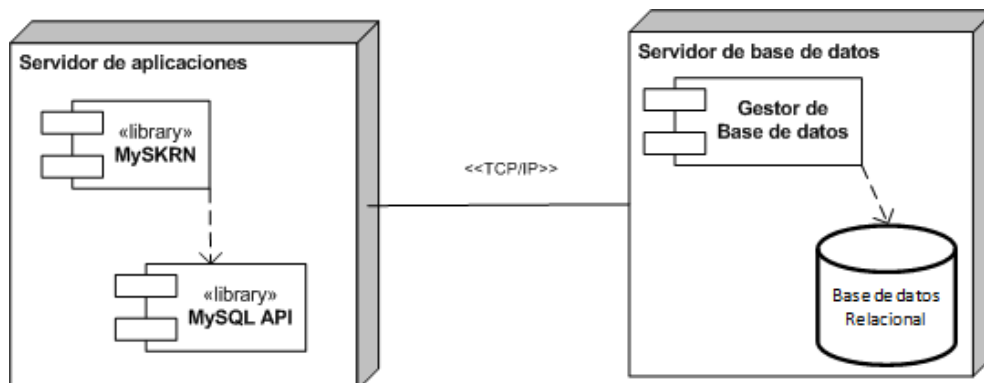
Figura 4. **Vista de despliegue del acceso a datos**



Fuente: elaboración propia con MS Visio 2007.

Por ser un patrón MVC, se tiene la capacidad de separar los datos de la lógica en servidores diferentes, proveyendo mayor independencia entre componentes. El acceso a datos es una subcapa de la capa de controlador en el patrón MVC, por lo tanto, su ubicación es dentro del servidor de aplicaciones.

Figura 5. **Vista física del acceso a datos y los datos**



Fuente: elaboración propia con MS Visio 2007.

En las siguientes secciones se describe con detalles los componentes de esta capa, su estructura lógica y de procesos, las operaciones incluidas y ejemplos de utilización.

### **3.1. Objetos transaccionales: MyBDItem**

Forma parte de la subcapa de acceso a datos en la capa de controlador del patrón MVC, se encarga de la inserción, actualización, eliminación, carga y transacciones de información, así también validación, sanación de datos y mensajes de error hacia el usuario. En la configuración se incluyen los datos utilizados para la conexión a la base de datos:

- Nombre o IP del servidor
- Usuario de conexión
- Contraseña de conexión
- Juego de caracteres
- Nombre de la base de datos
- Formato de obtención de datos fecha
- Formato de obtención de datos fecha-hora
- Mensajes de error

Cada objeto representa una fila de una tabla en la base de datos, a la cual se configura y la tabla que está relacionada al objeto es definido en el constructor, este es el momento donde se aprovecha la característica de PHP de que los objetos no necesitan tener variables de clase predefinidas, permitiendo que los campos de la tabla definida se obtengan de forma dinámica, reutilizando un mismo objeto hacia muchas tablas. Para que esto suceda la configuración debe tener acceso a los metadatos de la base de datos.

Utiliza una adaptación del patrón de diseño creacional Factory Method, se crea una clase con la configuración que hereda los métodos de la clase MyBDItem que es la clase abstracta constructora, la clase creadora en concreto, sería la nueva clase, que en este ejemplo se le pondrá SistemaBDItem.

Figura 6. **Vista lógica del uso del objeto transaccional**



Fuente: elaboración propia con MS Visio 2007.

El método creador es en este caso, el método constructor del objeto SistemaBDItem, donde se envía el nombre de la tabla a la cual se asocia dicho objeto.

```
$tabla1 = new SistemaBDItem('tabla1');
```

Como el patrón de diseño lo indica, la clase no provee la clase de objeto que se crea, pudo haber sido tanto 'tabla1', como 'tabla2' dependiendo del parámetro en el constructor y este objeto tendrá los atributos de la tabla asociada. Este mismo objeto SistemaBDItem permite crear toda la familia de objetos (en este caso, las tablas de la base de datos) sin especificar la clase, centralizando la creación de todos los objetos en una sola clase.



El archivo para almacenar en la ruta:

- <raíz-proyecto>/app/model

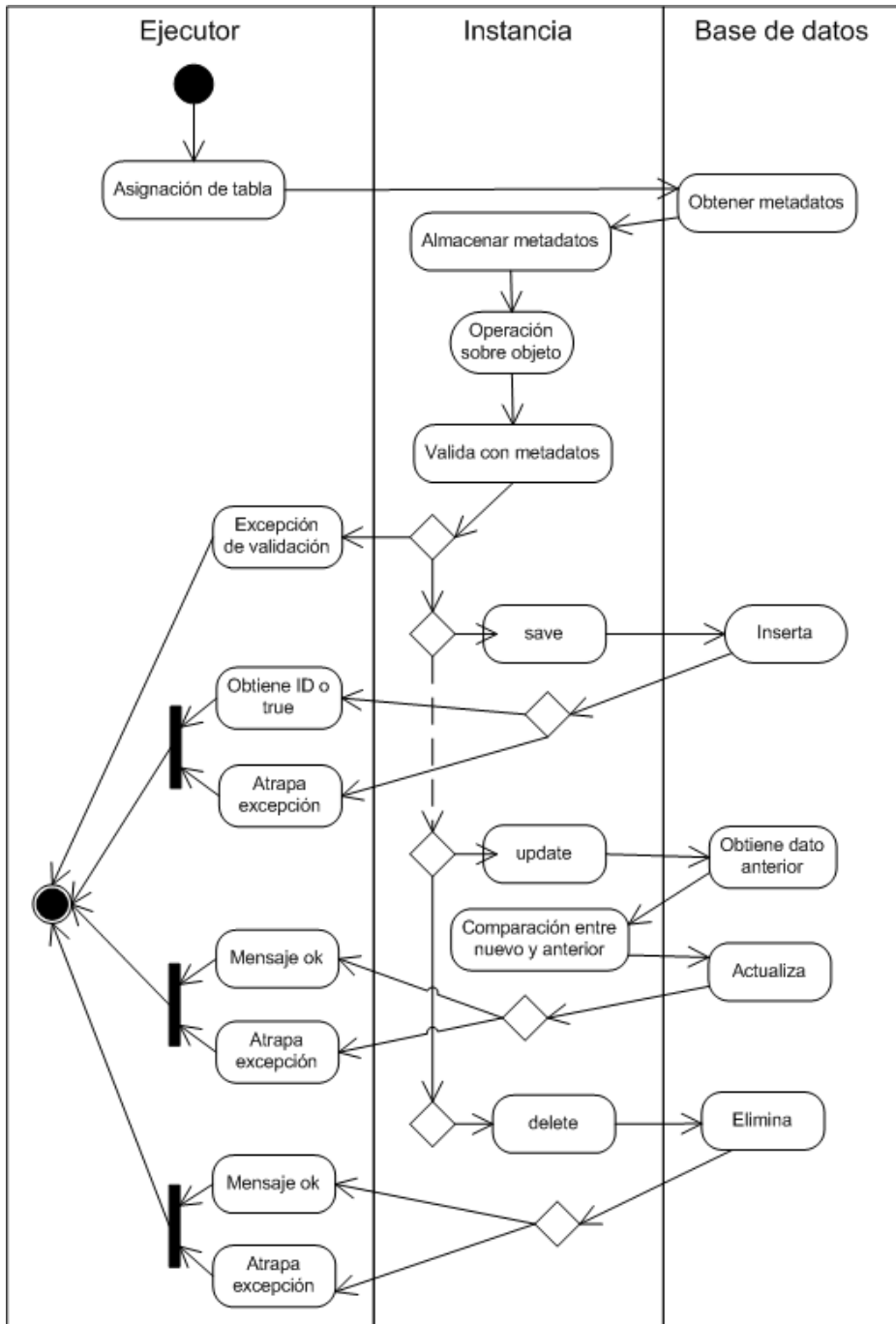
Para insertar en una tabla se utiliza el método save, para actualizar un dato se utiliza el método update y para eliminar se utiliza el método delete, todos están sobrecargado y los parámetros pueden ser:

Tabla I. **Parámetros del métodos de operación**

save	update	delete	Parámetros.
SI	SI	SI	Ningún parámetro.
SI	SI	SI	1. Arreglo de datos con los nombres de los campos en los índices.
SI	SI	SI	1. Arreglo de datos con los nombres de los campos en los índices. 2. Id de usuario.
SI	SI	NO	1. Arreglo de datos con los nombres de los campos en los índices. 2. Id de usuario. 3. Arreglo de campos que se obvian en la validación.
SI	SI	SI	1. Id de usuario.
SI	SI	NO	1. Id de usuario. 2. Arreglo de campos que se obvian en la validación.

Fuente: elaboración propia con MS Word 2007.

Figura 7. Vista de procesos de objetos transaccionales



Fuente: elaboración propia con MS Visio 2007.

### **3.1.1. Validación y sanación de datos**

Esta característica permite saber si un dato está en el formato correcto, si un campo es obligatorio y no está ingresado o definido, o si un dato está en un tamaño más grande de lo permitido por el campo. Si sucede alguna de estos eventos, antes de intentar almacenar, actualizar o eliminar, lanzará una excepción con el mensaje del error correspondiente.

La sanación de datos se utiliza contra los ataques más comunes en las aplicaciones, como la inyección de código SQL en las consultas o datos, o la inserción de código HTML y/o javascript dinámico, los objetos tienen la capacidad de eliminar comillas para prevenir inyección de consultas y eliminación de etiquetas HTML y código javascript de los datos.

A veces algunos datos son propio código HTML y/o javascript, cuando se almacena o actualiza se puede configurar el objeto para que omita los campos y no sean validados y/o sanados.

### **3.1.2. Mensajes de error y subclasses**

Cuando hay un error en los datos a insertar, actualizar o eliminar, se genera una excepción en el cual se incluye el mensaje de que fue lo que sucedió, estos mensajes son genéricos, pero pueden ser personalizados en la configuración del objeto, en cualquier lugar donde se haga una operación, este mensaje definido será el desplegado.

En algunos diseños de base de datos, una tabla contiene información que puede ser clasificada, por ejemplo, una tabla de usuarios en una base de datos de un colegio, esta puede contener la información de los alumnos, personal

docente, personal administrativo, entre otros, y cada uno de ellos debe tener un mensaje personalizado, el constructor tiene un segundo parámetro opcional, que es el nombre de la subclase a la que se hace referencia, por ejemplo, se usa la tabla usuario, esta se envía en el constructor, pero se quiere utilizar mensajes personalizados de alumno, este se enviara en el segundo parámetro para que pueda buscar el mensaje de error correspondiente.

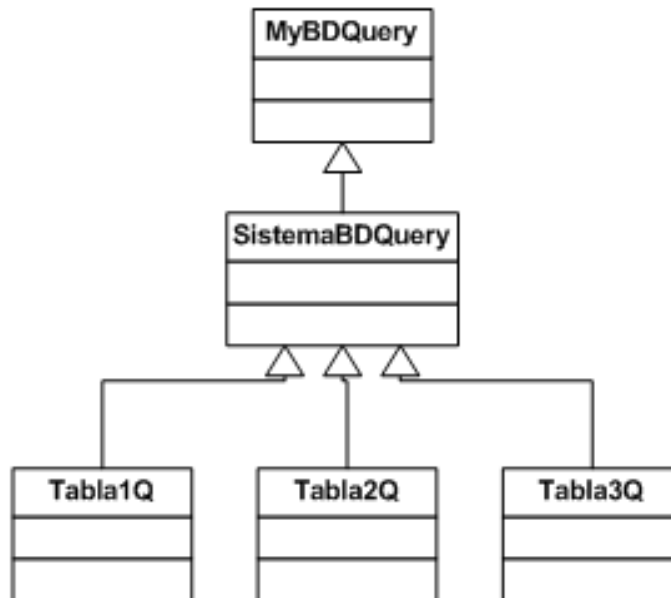
### **3.2. Objetos de consultas: MyBDQuery**

Forma parte de la capa de acceso a datos en el controlador del patrón MVC, se encarga de la ejecución de las consultas de las aplicaciones, con la opción de filtrar mediante condiciones que se arman dentro de la consulta a ejecutar, una vez ejecutada la consulta, se almacenan los resultados en objetos de arreglos para su manipulación en los archivos de control de la capa de control. Contiene cuatro operaciones para la extracción de datos mediante consultas:

- toDataSet: extrae un conjunto de datos dentro de una matriz de datos.
- oneRow: extrae una fila de datos y los almacena en un arreglo.
- howMany: extrae un campo de una fila de datos y lo almacena en una variable simple.
- toOptionList: extrae los datos y los coloca en una lista de un objeto select en una página HTML.

Para utilizar las consultas crea un objeto que extienda de MyBDQuery, en este se configuran los datos de la conexión a la base de datos, los mismos que en el objeto transaccional, luego esta será la superclase para las demás que contendrán las consultas, y dependiendo del requerimiento, se utilizará uno de los métodos descritos anteriormente.

Figura 8. Vista lógica del uso de los objetos de consulta

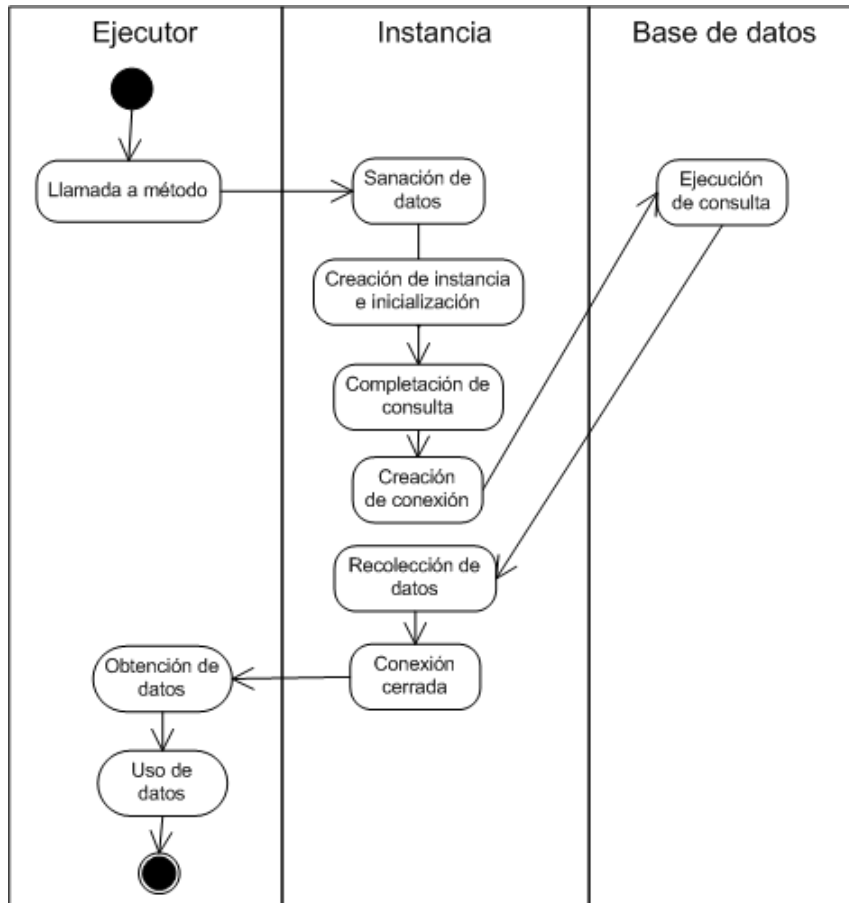


Fuente: elaboración propia con MS Visio 2007.

Los archivos de consulta y el de conexión se guardan dentro de la ruta

- <raiz-proyecto>/app/querys

Figura 9. Vista de procesos de objetos de consulta



Fuente: elaboración propia con MS Visio 2007.

### 3.2.1. Protección contra inyección SQL

Las consultas son el blanco de ataques de inyección SQL y este sucede, cuando el atacante utiliza las comillas para finalizar una cadena y luego el poder reescribir la consulta y extraer datos, este es prevenido mediante sanación de datos, los parámetros ingresados son sanados por un método que elimina las comillas y prevenir el ataque.

### **3.3. Bitácora de datos**

Una bitácora es una estructura que almacena un histórico de las acciones realizadas en una aplicación, en este caso la bitácora es sobre las operaciones sobre la base de datos. El uso de esta es opcional, puede ser activada o desactivada cuando sea requerido, como valor predeterminado, esta activada. Cuando se ejecuta una operación (save, update o delete), si no se envía el ID del usuario que ejecuta la acción, no se almacenara en bitácora, puesto que como no se sabe quien almaceno, no es necesario registrar esa acción.

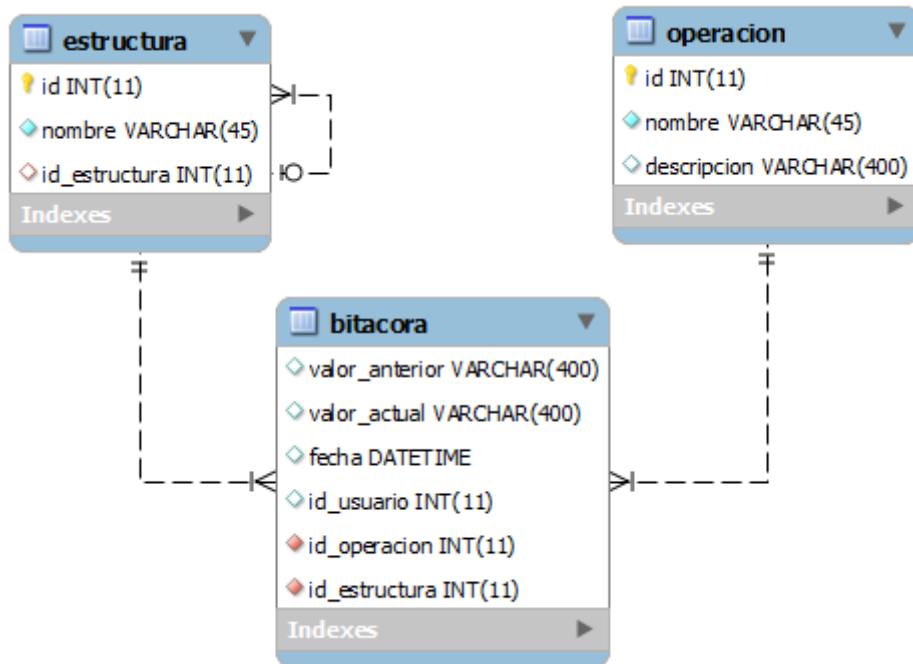
#### **3.3.1. Estructura**

La bitácora está formada por tres entidades:

- Estructura: almacena la estructura de la base de datos, desde el esquema, pasando por las tablas y los campos de cada tabla.
- Operación: almacena las operaciones que se realizan sobre la base de datos, normalmente son solamente Insertar, Eliminar y Editar.
- Bitácora: almacena todos los cambios realizados, el dato actual, el anterior, almacena el campo editado, la operación, la fecha y la hora, y la persona que realizó la acción.

Si por algún motivo las entidades no existen, se generan automáticamente, por consiguientes, la estructura de las tablas también lo hace en la entidad de estructura y por último se almacenan los cambios en la bitácora. Si las tablas fueran a tener en un futuro, cambios como eliminación y agregación de campos, estos son obviados o agregados a la estructura, según sea el caso, sin afectar el resto del funcionamiento.

Figura 10. Esquema E-R de la bitácora



Fuente: elaboración propia con MySQL Workbench 2.5 CE.



## 4. CAPA DE CONTROL

Forma parte de la capa de controlador en el patrón MVC, se encarga de ejecutar la lógica de la aplicación, realización de consultas, uso de los objetos de acceso a base de datos y ejecución algoritmos. Estos se ubicarán en la carpeta: <raíz-proyecto>/app/control.

Estos archivos de control deben heredar de una superclase llamada WebComm, del paquete SRKN-API, la implementación de esta clase conllevó a utilizar el patrón de diseño de comportamiento Strategy, se tiene un objeto transaccional (MyBDItem), equivalente al contexto del objeto, provee de las operaciones de manipulación de datos, explicadas en el capítulo anterior y se implementan mediante los métodos:

- Insert: inserta un dato en la base de datos.
- InsertID: inserta un dato en la base de datos y si la tabla tiene la llave primaria auto incrementable, regresa el valor de la llave primaria generada.
- Update: actualiza una fila en una tabla de la base de datos en base a su llave primaria o candidata.
- Delete: eliminar una fila en una tabla de la base de datos en base a su llave primaria o candidata.
- Load: carga una fila de una tabla de la base de datos en base a su llave primaria o candidata en un XML, y este tiene el nombre de los campos en cada miembro del XML, útil cuando se requiere un acceso rápido a los atributos.

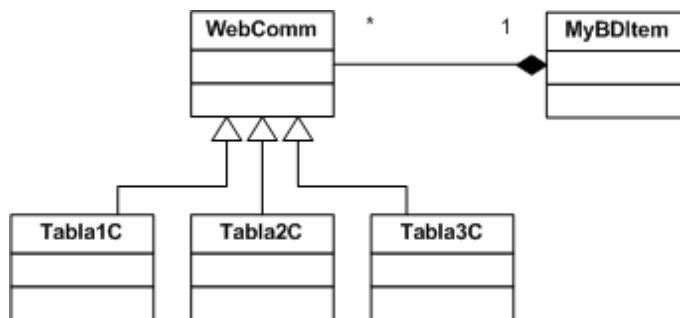
- Load2: carga datos de una fila en una tabla de la base de datos en base a su llave primaria o candidata en un XML, pero los campos tienen su propio miembro, y el nombre del campo es parte del atributo del miembro, útil cuando se requiere cargar la información en un formulario en HTML.

Así mismo, hereda dos variables para el manejo de los datos enviados desde la vista:

- \_\_data: arreglo de datos enviados desde la vista
- \_\_webuser: identificador del usuario en sesión

Para usar los métodos Update, Delete, Load y Load2 es necesario conocer el valor de las llaves primarias o candidatas de la fila para su realización. Si no lo lleva, lanza un MyBDEException que hay que manejar. De igual manera, excepciones como duplicidad de llaves, campos requeridos, llaves foráneas inválidas y demás, son manejados en la capa de control. El siguiente diagrama muestra las clases con la implementación del patrón de diseño strategy.

Figura 11. **Vista lógica del diagrama de clases**



Fuente: elaboración propia con MS Visio 2007.

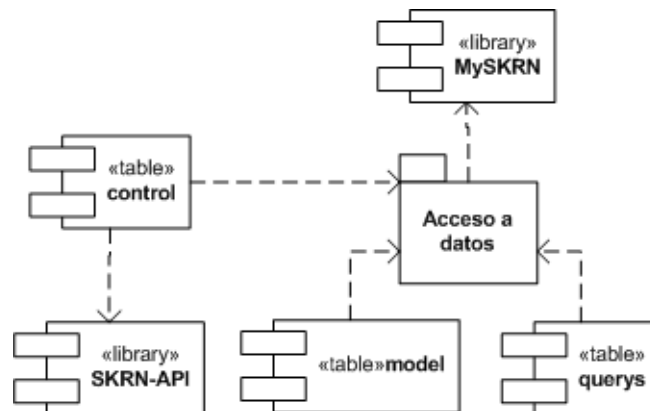
Se requiere que en el constructor de la clase de control inicialice dos variables de clase que hereda de WebComm:

- \_\_tabla: nombre de la tabla sobre la cual se realizan las operaciones de mantenimiento.
- \_\_objeto: nombre de la clase de objetos transaccionales.

Como es de suponer, se debe usar un archivo de control por cada tabla que se utilice en la base de datos, pero no es obligatorio utilizar esta característica, los métodos de mantenimiento pueden ser reescritos si es necesario.

En esta capa se incluyen la mayoría de clases del paquete SKRN-API, como el DataGrid2 para la generación de tablas de datos, Calendar, para la generación de calendarios, Utiles, que contiene métodos utilitarios y auxiliares. El siguiente diagrama muestra cómo están los componentes relacionados, el acceso a datos, el control y las librerías que se implementan.

Figura 12. **Vista de despliegue de la capa de control**



Fuente: elaboración propia con MS Visio 2007.

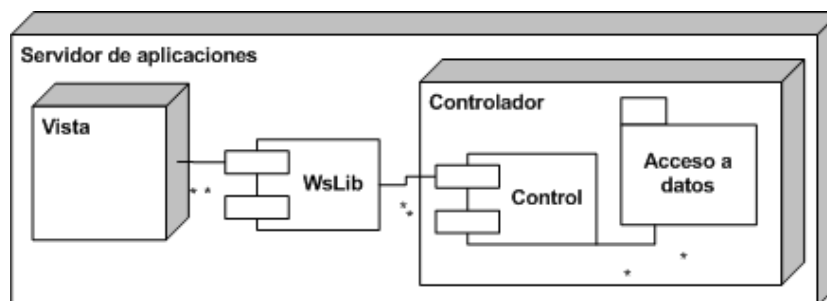
#### 4.1. Interfaz de exposición de clases y métodos: WSLib

Esta interfaz se encarga de exponer los archivos de control hacia la capa vista de la aplicación web, conectándose para enviar y recibir información desde los archivos de interfaz. Cabe mencionar que, pueden conectarse mediante distintos modos, pero deben regresar datos del mismo tipo en caso de que se migre de un tipo de conexión a otra, como convención, los archivos de control envían cadenas de caracteres, tantos mensajes XML, texto simple y en el caso de binarios, estos codificados en Base 64 para luego ser decodificados del lado de la vista.

##### 4.1.1. Conexión por instancia

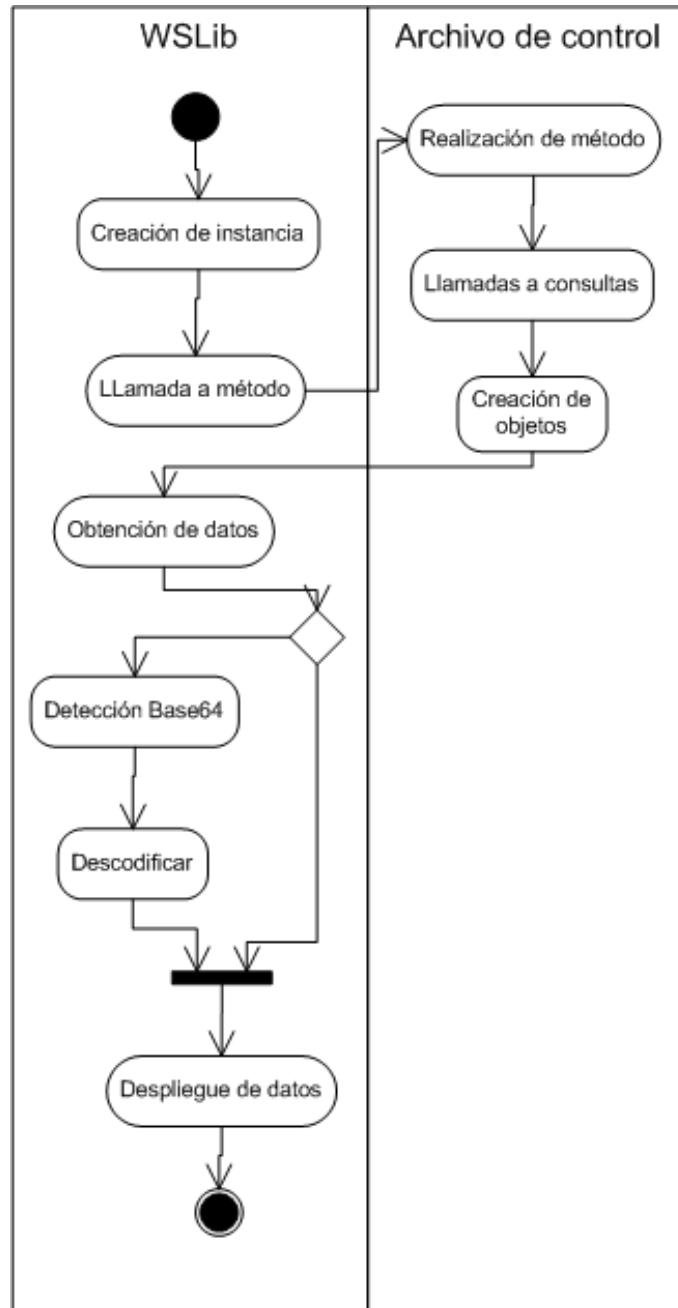
Si la capa de controlador y el de vista están en el mismo servidor, se utiliza este tipo de conexión. La interfaz crea una instancia al recibir el nombre de la clase, el nombre del método, los posibles parámetros que se deseen enviar y el identificador del usuario conectado a los archivos de control, de este modo no es necesario incluir todos los archivos de control para hacer llamadas a los métodos.

Figura 13. Vista física de la conexión por instancia



Fuente: elaboración propia con MS Visio 2007.

Figura 14. Vista de procesos de la conexión por instancia

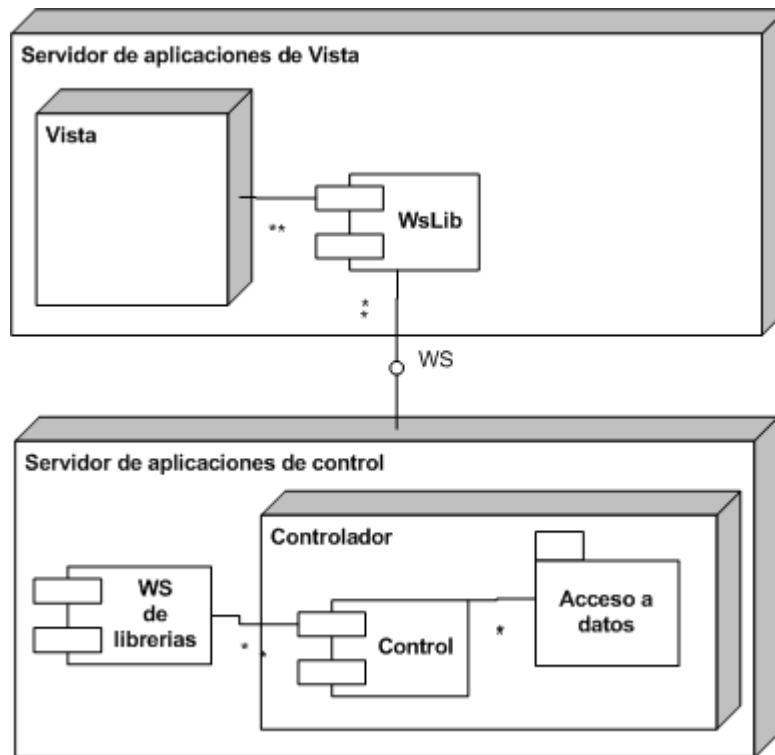


Fuente: elaboración propia con MS Visio 2007.

#### 4.1.2. Conexión por servicio web

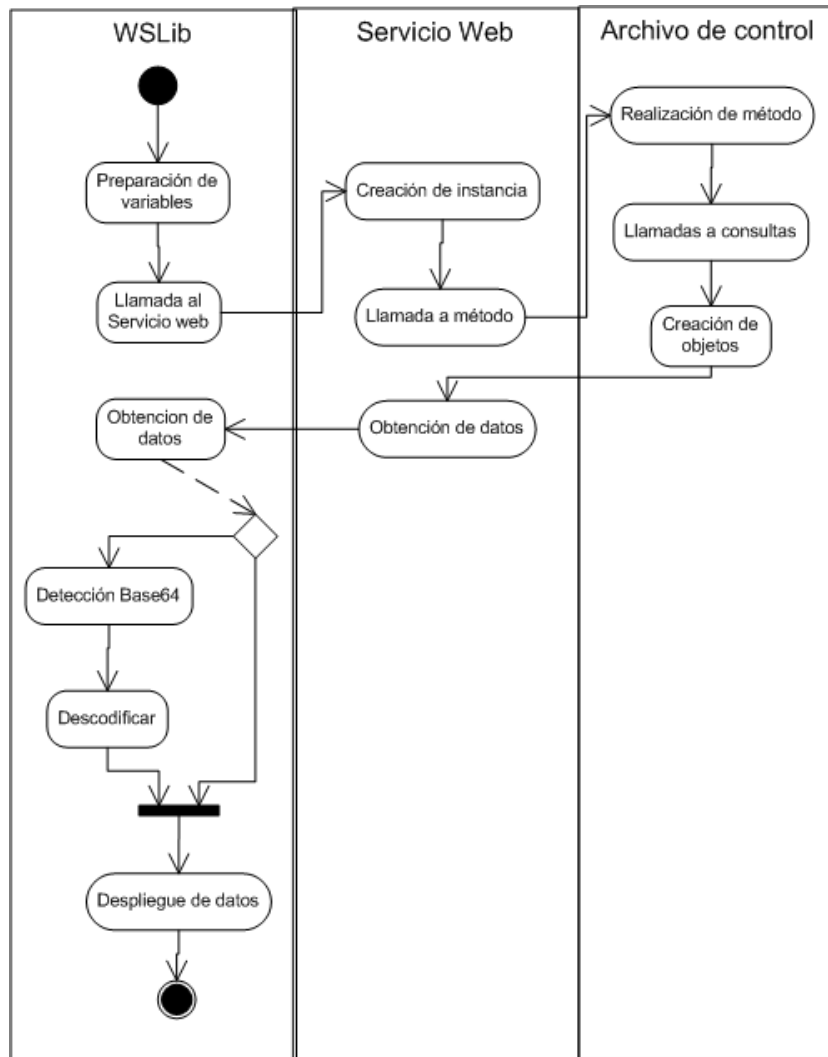
Si la capa de controlador y de vista está en distintos servidores, estos usan este tipo de conexión. Se envían los datos de igual forma que la conexión por instancia, pero en este caso el servicio web crea la instancia y ejecuta el método, luego envía los datos a la capa de vista por servicio web, la clase WSLib se vuelve parte de la capa de vista pues este envía y recibe los datos del servicio web.

Figura 15. Vista física de la conexión por servicio web



Fuente: elaboración propia con MS Visio 2007.

Figura 16. Vista de procesos de conexión por servicio web



Fuente: elaboración propia con MS Visio 2007.

#### 4.2. Librerías auxiliares

Son herramientas seleccionadas por su alta cohesión y bajo acoplamiento en las aplicaciones que implementan la arquitectura.

#### **4.2.1. SKRN-API**

Es una librería que contiene múltiples clases utilitarias, interfaces a servicios web, exposición de métodos específicos del sistema de roles y permisos que se describirá posteriormente, contiene clases para la generación de tablas de datos, calendarios, entre otras.

#### **4.2.2. Generador de gráficas: JPGraph**

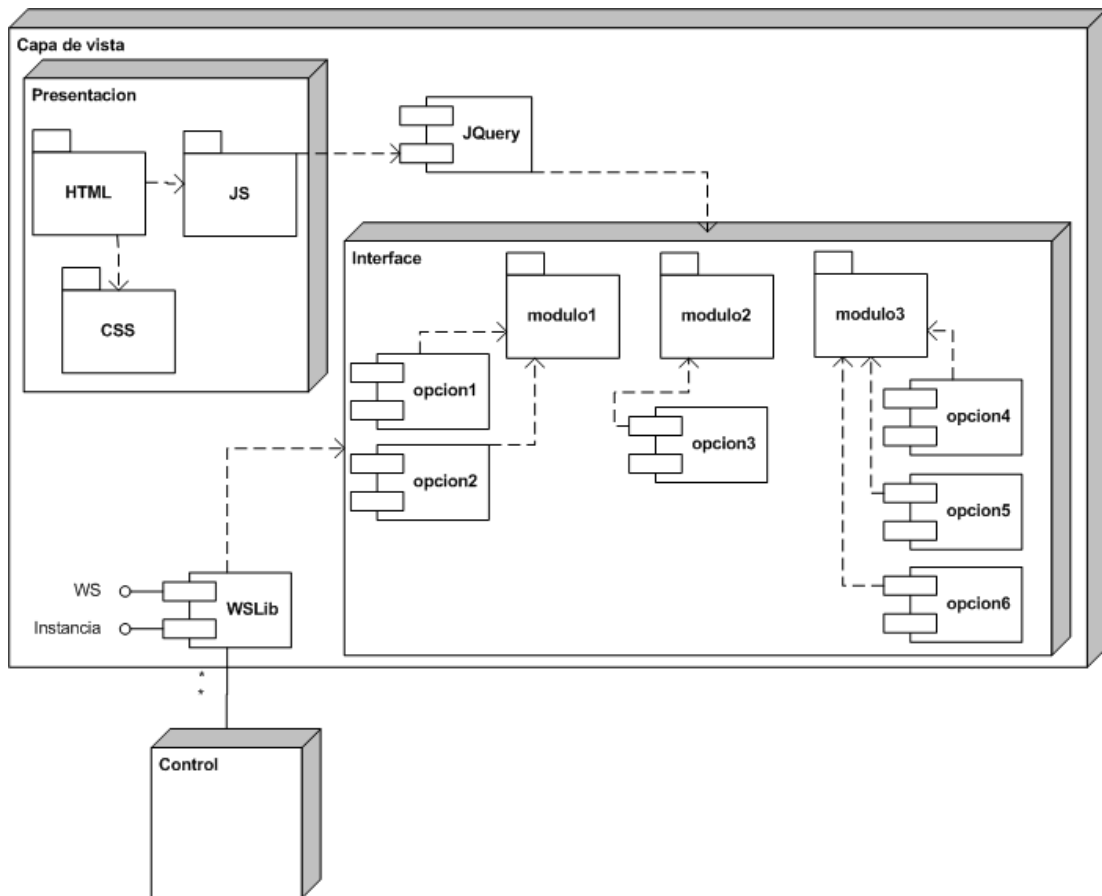
Es una librería encargada de generar gráficos, mayormente usados para representaciones de estadísticas en gráficas de pie, barras, puntos, etc.



## 5. CAPA DE VISTA

Se encarga del despliegue de formularios, tablas de datos, obtención de datos, manejo de sesiones y accesos de seguridad. Cada aplicación tiene módulos, que son unidades lógicas que agrupan las vistas del sitio web, a estas vistas se le llaman opciones.

Figura 17. Vista de despliegue

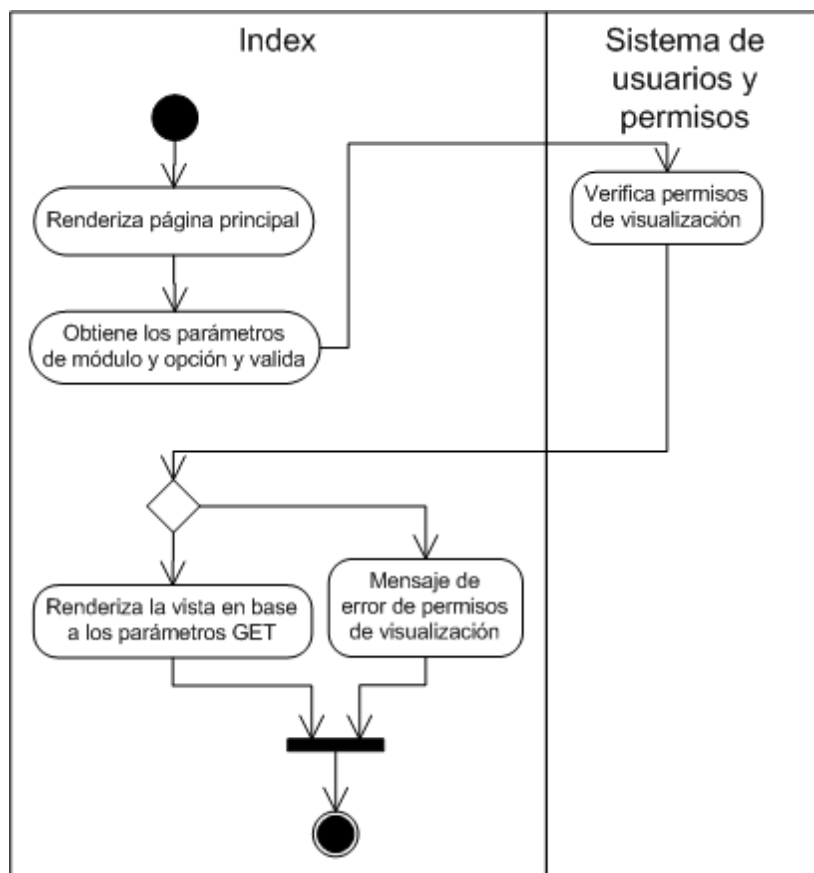


Fuente: elaboración propia con MS Visio 2007.

## 5.1. Organización de la presentación

La vista está dividida en dos partes, una página maestra que contiene el estilo, scripts, la estructura general, los menús de acceso a las páginas de las aplicaciones y pequeñas páginas llamadas vistas, con su propia funcionalidad específica de las opciones de las aplicaciones, con su propio script de funcionamiento y si lo requiere, su propio estilo.

Figura 18. Vista de procesos de una página



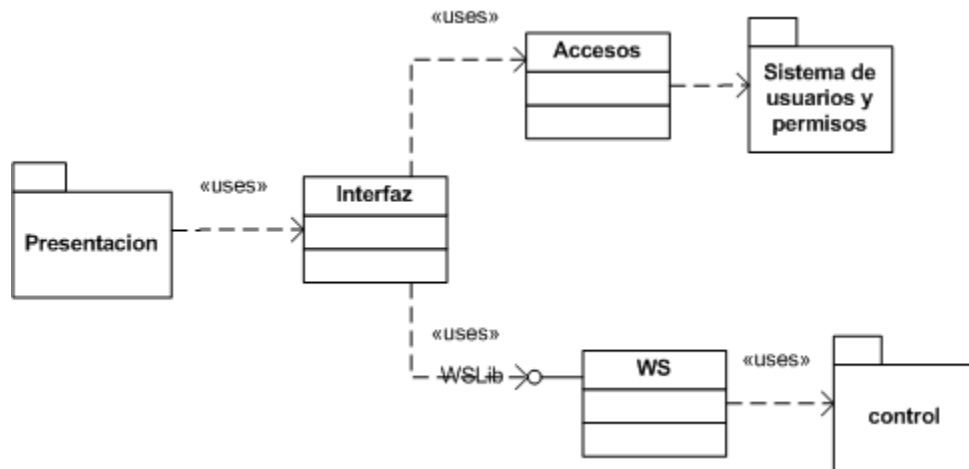
Fuente: elaboración propia con MS Visio 2007.

Las páginas están contenidas en una carpeta de vistas en la ruta

- <raíz-proyecto>/piece/

Dentro de esta se encuentran carpetas denominadas módulos, que contienen las vistas de páginas que son las opciones de la aplicación. La siguiente grafica muestra como se relacionan los distintos componentes de la capa de vista.

Figura 19. Vista lógica de la capa de vista



Fuente: elaboración propia con MS Visio 2007.

El método del manejo de sesiones debe iniciarse en todas las páginas donde se requiera el uso de la misma, con este manejo de vistas solo se necesita incluirla en la página maestra.

Si son necesarios métodos y archivos en javascript para funciones generales, se pueden colocar en la página maestra para que todas las páginas

que lo requieran, puedan utilizarlos y solo incluirla una vez en el código, si se necesitan métodos específicos, se incluyen solamente en cada vista por separado, minimizando así, la complejidad del código en javascript.

## **5.2. Javascript y archivos de interfaz**

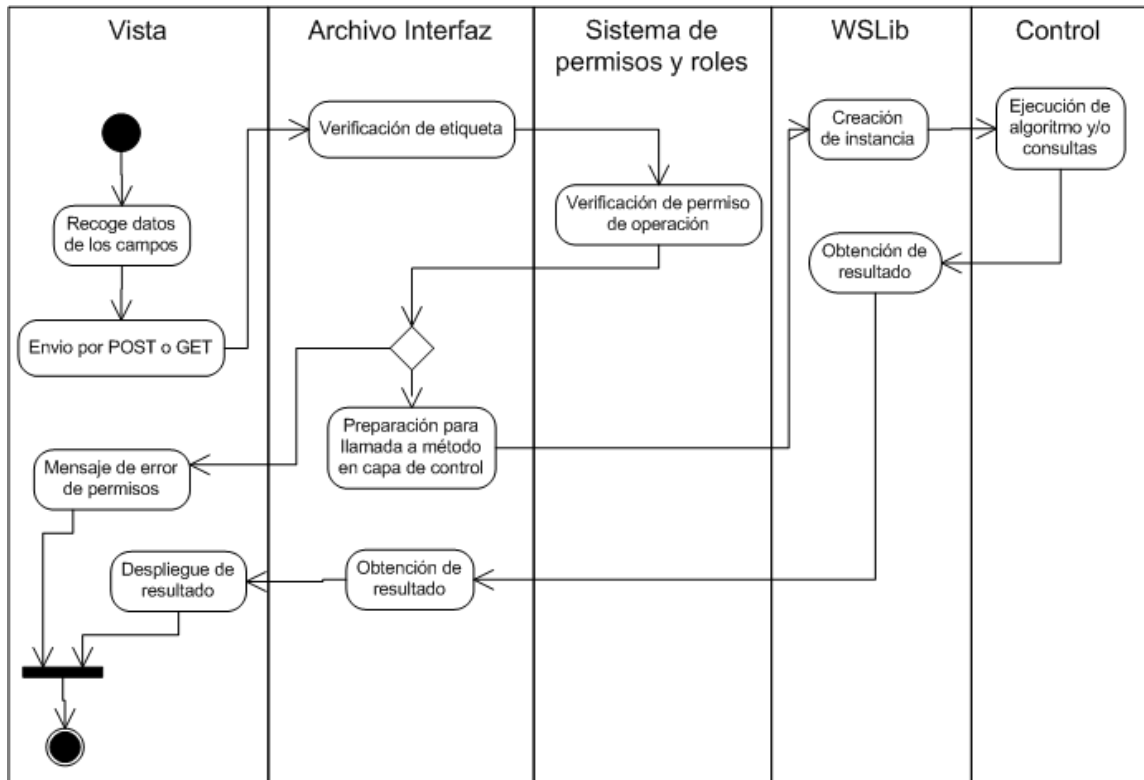
El código HTML de cada pieza de las opciones del sistema está compuesto exclusivamente por codificación HTML, a su vez este asocia las acciones a un archivo extensión JS con todas las instrucciones javascript para su funcionamiento, a veces si se necesita un estilo extra, se utiliza un archivo extensión CSS para los estilos de las páginas. La organización de los archivos de vista, javascript y css es:

- Vistas: <raíz-proyecto>/piece/modulo/opcion.phtml
- Javascript: <raíz-proyecto>/js/modulo/opcion.js
- CSS: <raíz-proyecto>/css/modulo/opcion.css

Como convención de nombres el nombre de la vista, es igual al nombre del archivo javascript y del CSS, de igual modo el nombre del módulo donde está contenido el archivo.

La vista debe comunicarse con el controlador y lo hace por medio e unos archivos de interfaz, estos se encargan de recoger la información y la envían hacia el controlador, usando como parámetros, el nombre de la clase, el método a ejecutar, los parámetros enviados dentro de un arreglo y si es requerido, el identificador del usuario. El siguiente diagrama muestra como se comunica la capa de vista con la de controlador mediante conexión por instancia.

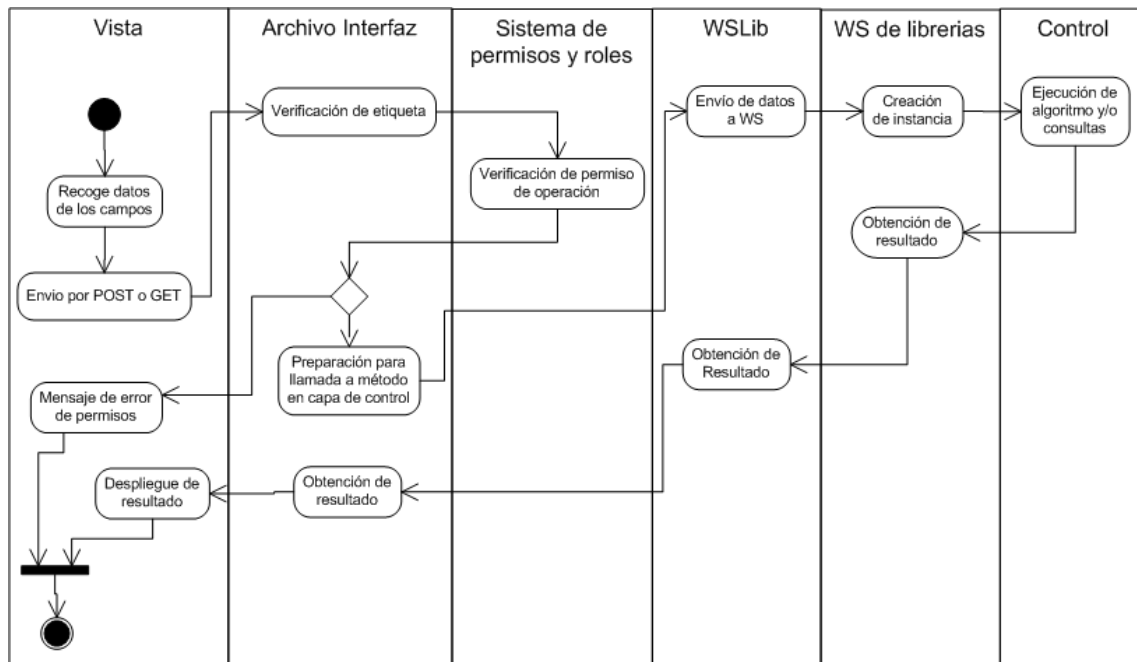
Figura 20. **Vista de procesos de enviar y recibir información**



Fuente: elaboración propia con MS Visio 2007.

El código javascript recoge la información de los formularios y las envía por medio de los métodos POST o GET según sea el caso, a los archivos de interfaz, estos recogen el arreglo de datos y crean la instancia de la clase definida en la llamada y al método a ejecutar. Ya dentro del método, este se encarga del manejo de los parámetros enviados por los métodos POST o GET. El siguiente diagrama muestra como se comunica la capa de vista con la de controlador por medio de conexión de servicio web.

Figura 21. Envío y recuperación de información con WS



Fuente: elaboración propia con MS Visio 2007.

A su vez, cuando el controlador envía una respuesta, el archivo de interfaz, recoge la información y la envía al javascript para que este lo interprete (en el caso de mensajes XML) y lo despliegue en el HTML. Si fuera un binario, este vendrá en codificado en formato base 64, la interfaz tiene la capacidad de transformarlo en un binario al descodificarlo a su formato binario original y descargarlo o mostrarlo.

### 5.3. Sesiones

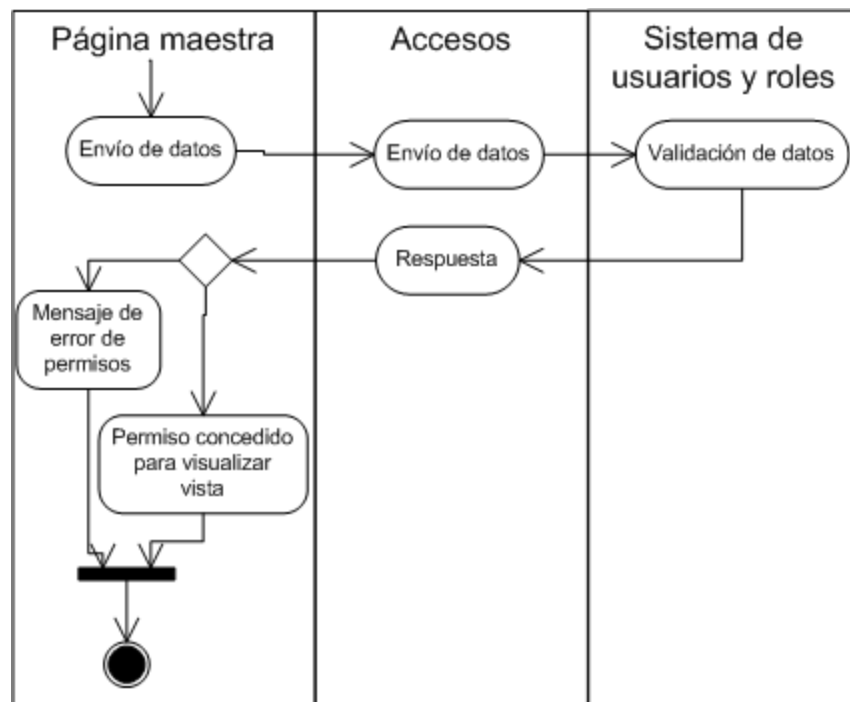
Las sesiones en la arquitectura se manejan de forma automática por el lenguaje PHP. Solamente se utiliza una página de ingreso para la recolección

de credenciales y la validación de estas, por medio de la base de datos del Sistema de usuarios y permisos que se verá más adelante.

### 5.3.1. Permisos de visualización

Cuando un usuario ingresa al sistema, este tiene roles asociados, los roles definen que páginas puede visualizar y cuáles no, esto se decide por medio de la URL de la opción a la cual se desea entrar, el sistema verifica si el usuario tiene permisos para visualizar la URL escrita y decide mostrarla o no.

Figura 22. Vista de procesos. Permiso de visualización

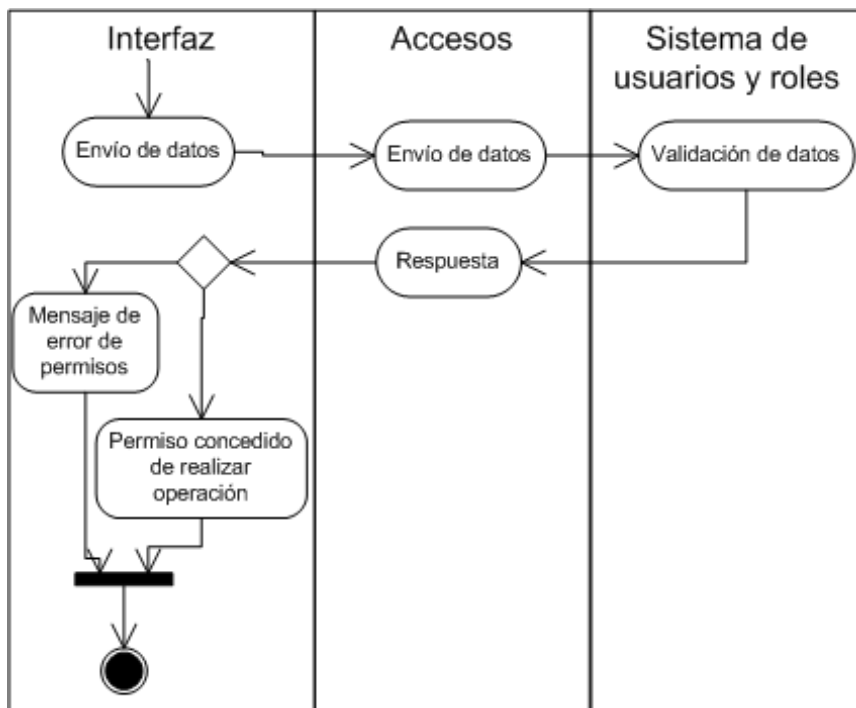


Fuente: elaboración propia con MS Visio 2007.

### 5.3.2. Permisos de operación

Cada opción tiene también operaciones, que son Listar, Actualizar, Eliminar, Listar, Reporte, cada rol tiene permisos de ejecutar estas operaciones, y se definen en las interfaces junto a la operación cuando se va a llamar. Si el rol no tiene permisos para ejecutar la operación, simplemente enviara mensaje de error.

Figura 23. Vista de procesos. Permiso de operación



Fuente: elaboración propia con MS Visio 2007.



#### **5.4. Ayuda en línea**

Las opciones tienen distintas funcionalidades que dependen del requerimiento que se haya entregado, la vista tiene una opción de ayuda en línea, es un panel que muestra una descripción de que funcionalidades tiene la vista y las resalta al elegir el nombre de un componente en el mismo panel.

#### **5.5. Librerías Javascript**

Se provee de un script maestro, común en todas las aplicaciones que implementen la arquitectura, provee de funciones que facilitan la integración de la capa de control con la capa de vista, así como numerosos componentes javascript para la visualización de tablas, calendarios, etc. Las librerías son:

- master: conjunto de funciones comunes para la manipulación de información.
- skrn-datagrid2: librería que muestra un listado de datos, tiene opciones para generar eventos, abrir links, entre otros.
- skrn-calendar: librería que genera un calendario para ordenar datos cronológicamente.
- skrn-tab: librería que genera un panel de pestañas para el ordenamiento de formularios.



## **6. SISTEMA DE USUARIOS Y PERMISOS**

En este capítulo se presenta el sistema encargado de almacenar los datos de los sistemas, su estructura, usuarios, roles, permisos de roles, generación de manuales.

### **6.1. Estructuración de sistemas**

Los sistemas en la arquitectura están formados por la siguiente jerarquía de componentes.

#### **6.1.1. Módulo**

Son estructuras que agrupan opciones intentando proveer alta cohesión, es decir, las funcionalidades de las opciones tienen un mismo fin, pero esto ya depende de quién diseñe el sistema mismo. La información de un módulo debe ser su nombre, su descripción, un alias y a qué sistema pertenece.

#### **6.1.2. Opción**

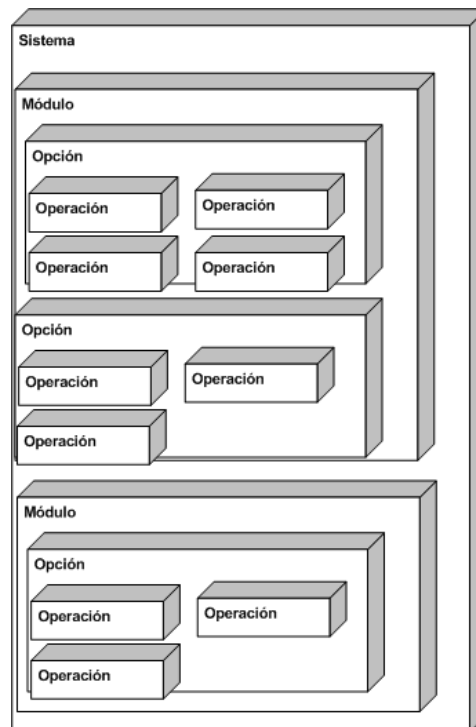
Son estructuras que equivalen a las vistas de las aplicaciones, tienen un nombre, descripción, un alias, la URL a la que apuntan, el tipo de aplicación que es, si está obsoleta o vigente y el módulo al que pertenecen.

### 6.1.3. Permisos

Son las unidades lógicas que asocian los sistemas con los roles, pero que además definen las operaciones que se pueden hacer sobre una opción, las operaciones son:

- Insertar
- Editar
- Eliminar
- Listar
- Registrado

Figura 24. Vista gráfica de un sistema



Fuente: elaboración propia con MS Visio 2007.

Por ejemplo, si una opción es para el mantenimiento de usuarios como altas bajas y cambios, esta opción en su archivo de interfaz tendrá las operaciones Insertar, Editar, Eliminar y Listar y los permisos asociados a estas llamadas.

## **6.2. Asignación de permisos a roles**

Los roles son estructuras que agrupan usuarios y dan permisos para entrar a las distintas páginas de la aplicación. También en base a estos, se genera el menú de opciones para que el usuario pueda acceder a las vistas de la aplicación.

Los roles están asociados a los sistemas mediante los permisos, que es la parte más granular de los sistemas, se este modo los roles pueden tener permisos altamente específicos para las distintas operaciones en la aplicación.

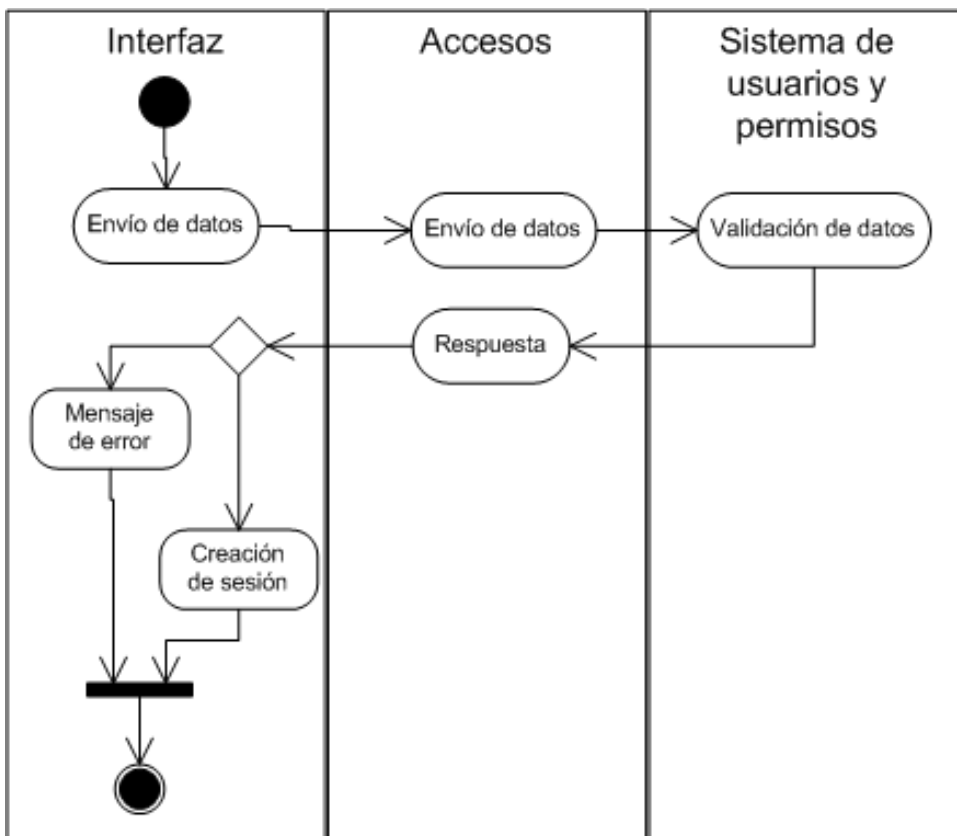
## **6.3. Usuarios**

Son las entidades que usan los sistemas. Estos necesitan almacenar credenciales para el ingreso a los sistemas, estas credenciales normalmente están en cada sistema a los cuales estos pertenecen, pero a mayor número de sistemas, mayor cantidad de credenciales, además de que para cada sistema se tendría que desarrollar un gestor de usuarios. Al condensar esta funcionalidad, se factoriza y desarrollo de este módulo ya no es necesario, los sistemas ahora solo necesitan una interfaz que permita el acceso a los datos. Esto se provee en el paquete SKRN-API con los métodos necesarios para:

### 6.3.1. Autenticación

Cuando un usuario desea ingresar a la aplicación, este ingresa su nombre de usuario y contraseña, la contraseña es encriptado por el algoritmo MD5 del lado del formulario por javascript y luego este es enviado y validado por el sistema.

Figura 25. Vista de proceso de autenticación

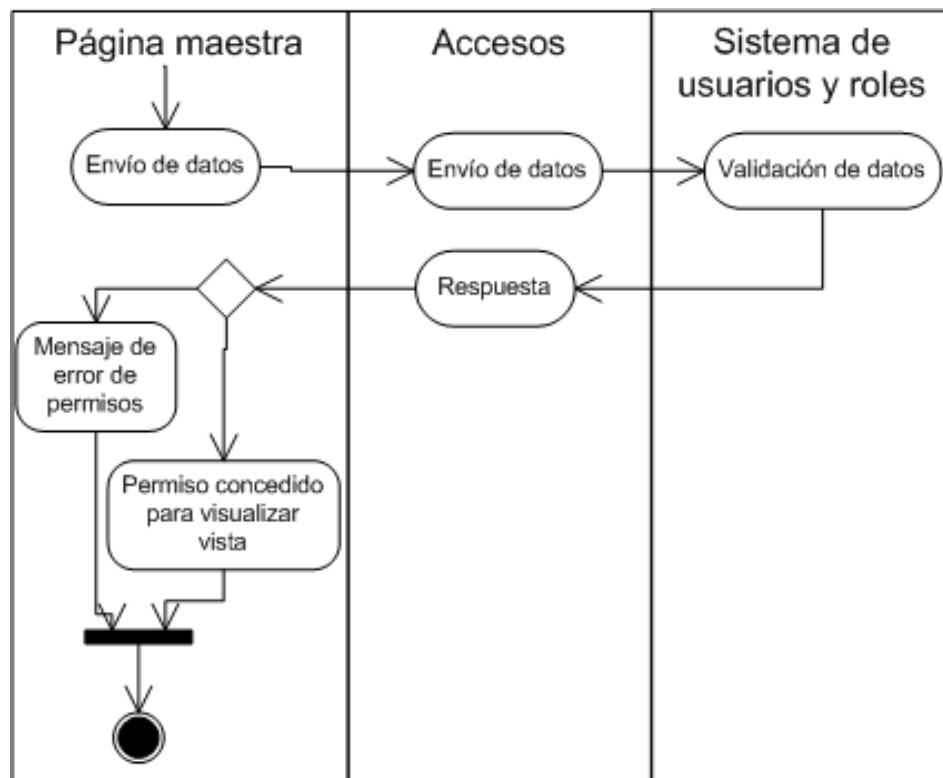


Fuente: elaboración propia con MS Visio 2007.

### 6.3.2. Verificación de permisos de visualización

Las vistas están asociadas a una URL y esta es única, se envía esta y el identificador e usuario y se verifica dentro del sistema, si alguno de los roles a los cuales el usuario está asociado, tiene asignado algún permiso de la vista, esta se desplegará, de lo contrario, dará un mensaje de error.

Figura 26. Vista de proceso de permisos de visualización

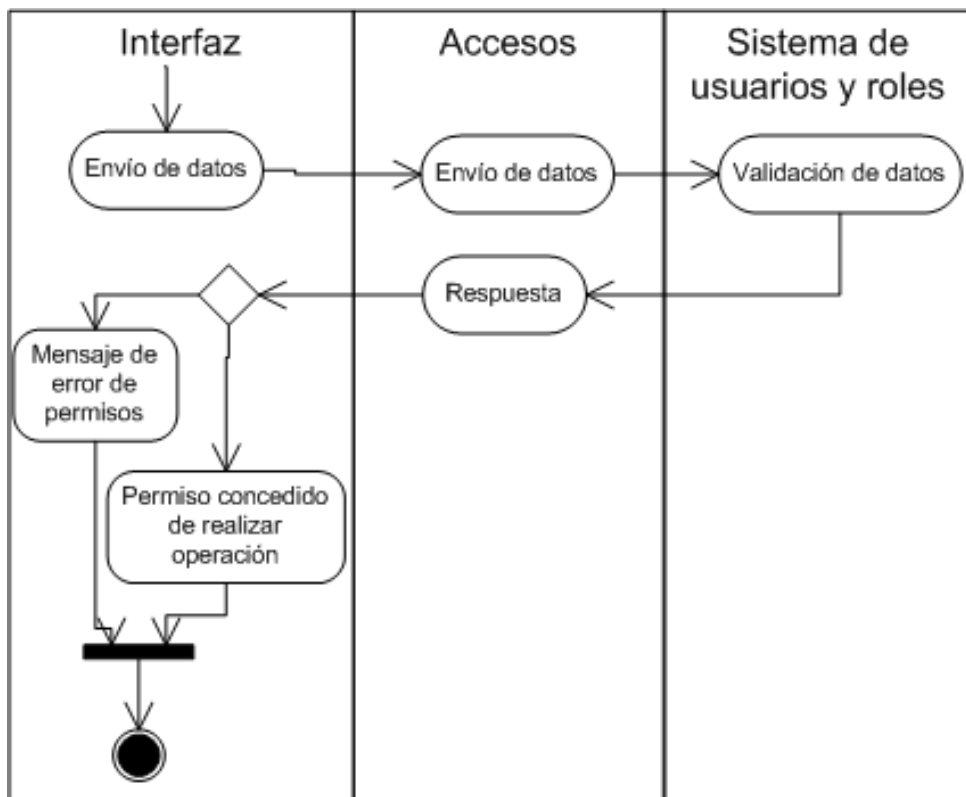


Fuente: elaboración propia con MS Visio 2007.

### 6.3.3. Verificación de permisos de operación

Las vistas tienen operaciones, y los roles se asocian a estas operaciones, pero no todos los roles tienen los mismos permisos, algunos solo pueden visualizar datos, otros pueden insertar, actualizar o eliminar. Los permisos de operaciones controlan este flujo de información. Si un usuario va a insertar datos, el sistema verifica que este tenga el rol indicado, si no, mostrará un mensaje de error.

Figura 27. Vista de procesos de permisos de operación



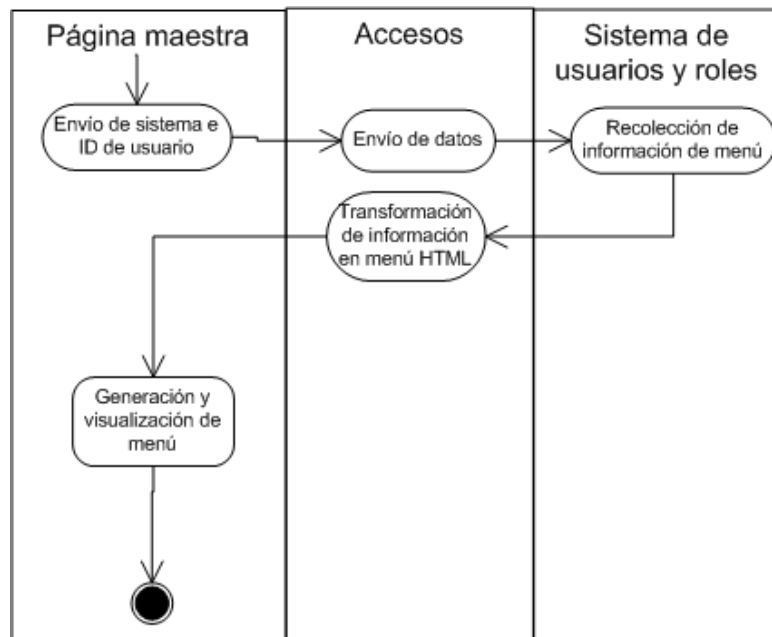
Fuente: elaboración propia con MS Visio 2007.



### 6.3.4. Obtención de menú

Los roles están asociados a las operaciones, así como estas a las opciones y módulos, la vista envía que sistema está siendo ejecutado y junto al identificador del usuario, verifica los roles que este tiene sobre el sistema y en base a esas asociaciones genera un menú de forma automática, con los vínculos dirigiendo a las opciones que el usuario puede ver.

Figura 28. Vista de procesos de obtención de menú



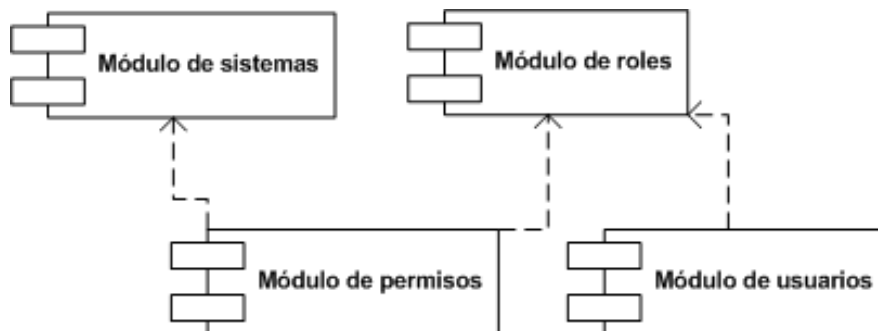
Fuente: elaboración propia con MS Visio 2007.

### 6.3.5. Carga de datos de sesión

La información almacenada en el sistema de usuarios y permisos es básica para el ingreso a los sistemas, ejecución de operaciones, envío de correos, etc, es importante que esta información esté cargada en sesión, dicha

información es obtenida cuando el usuario ingresa al autenticarse en el sistema. El siguiente diagrama muestra como se relacionan los módulos del sistema de usuarios y permisos.

Figura 29. **Vista de despliegue del sistema de usuarios y permisos**



Fuente: elaboración propia con MS Visio 2007.

### 6.3.6. **Sistemas en otros paradigmas**

Si hubiera sistemas que ya han sido desarrollados en otros paradigmas de programación incluso fuera del patrón de la arquitectura, los métodos están disponibles para ser implementados en estos sistemas.

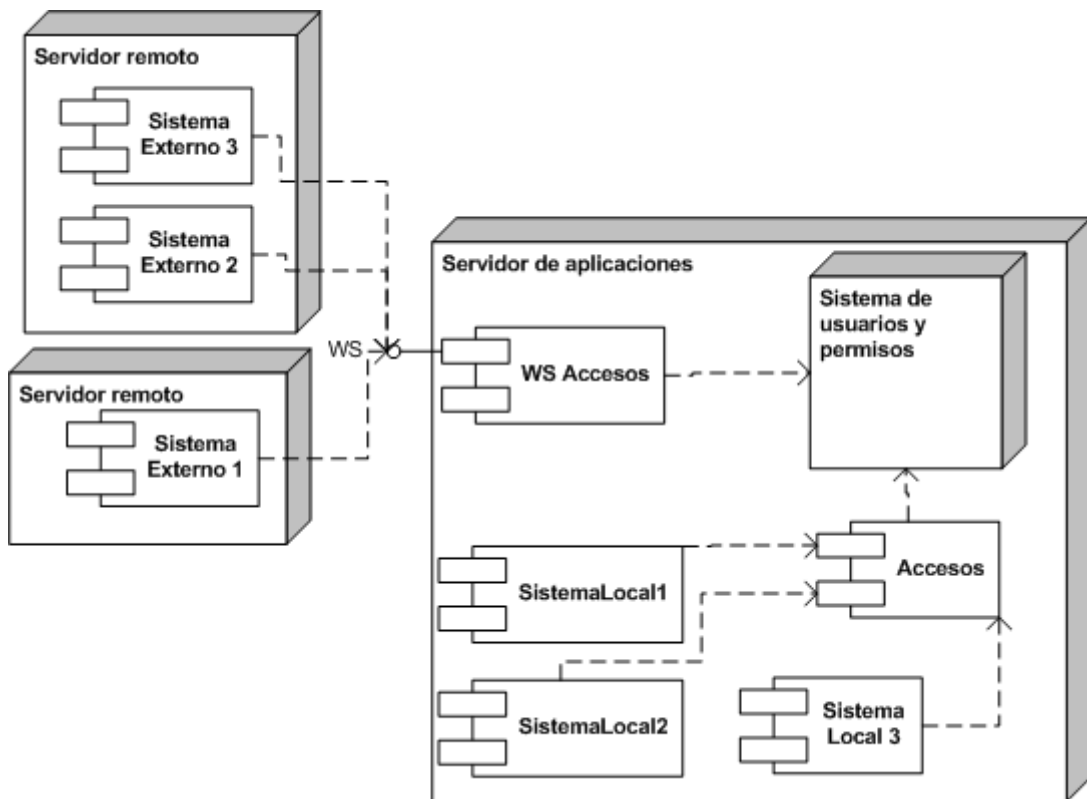
### 6.3.7. **Sistemas en servidores remotos**

Si hubiera sistemas que están ubicados en otros servidores, se provee de un servicio web que implementa las funcionalidades anteriormente mencionadas, para así, integrar sistemas remotos al sistema de usuarios y permisos.

### 6.3.8. Aplicaciones de escritorio

Si hubiese aplicaciones que son parte de un sistema, pero son de escritorio, estos usan el servicio web que se provee para los sistemas remotos. Ya dependiendo del lenguaje, el desarrollador debe encontrar la manera para conectar la aplicación con el sistema de usuarios y permisos por medio del servicio web.

Figura 30. Vista física



Fuente: elaboración propia con MS Visio 2007.

## **6.4. Documentación de sistemas**

Las vistas de los sistemas están divididas en paneles, los cuales contienen los formularios, tablas e imágenes. Se provee de un módulo para el ingreso de la descripción de las vistas y cada panel.

### **6.4.1. Widget de ayuda**

Es un objeto en la pantalla de la vista, normalmente se ubica en la parte superior izquierda de la pantalla. Se accede a él al solo darle un clic y desaparece de la misma manera. Cada vez que se cambie de panel en la vista de la aplicación, se refrescará la información contenida de la documentación.

Si la aplicación no usa el sistema de vistas de la arquitectura se puede adaptar, pero eso depende del programador y su ingenio.

### **6.4.2. Generación de manuales de usuario**

La información que aparece en el *widget* de ayuda es la misma que se utiliza para la generación de manuales de usuario, se realizó de este modo para que la documentación sea consistente e igual en cualquiera sus dos presentaciones, el documento se puede generar en formato PDF que es un tipo estándar para la visualización de documentos.

Dependiendo del rol, son las vistas que el usuario puede ver. El módulo de documentación provee la generación de manuales de usuario dependiendo del rol, de este modo, vistas que no necesita un usuario, no se incluirán en la documentación.

### **6.4.3. Generación de manuales técnicos**

La información de los sistemas es almacenada en este, por lo tanto, tiene la información de las conexiones a la base de datos, así también la información de la descripción de los sistemas, módulos, opciones y permisos, los roles y su asociación, la documentación técnica que se genera es:

- Diccionario de datos
- Información de los roles y permisos asociados
- Información de la estructura de los sistemas

## **6.5. Despliegue de capas de arquitectura**

La siguiente sección muestra las diferentes configuraciones de cómo la arquitectura puede ser desplegada en los servidores que se tenga disponibles.

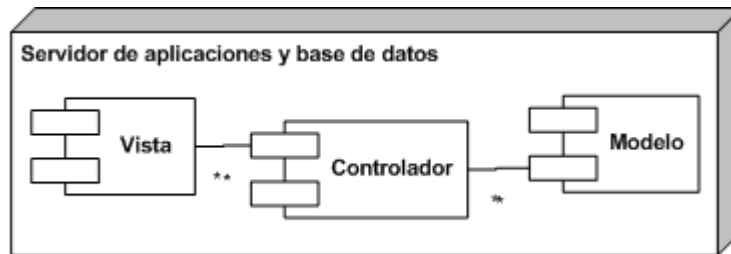
### **6.5.1. Un servidor**

Es la configuración más sencilla. Se cuenta con un solo servidor de aplicaciones y de base de datos, este contendrá todas las capas en el mismo.

- Ventajas
  - Se utilizan llamadas por instancia, la comunicación entre las capas es rápida.
  - El seguimiento de errores de software es más sencillo al tener todas las bitácoras de error en un solo lugar.

- Desventajas
  - Alto acoplamiento de capas.
  - La carga de trabajo del servidor, tanto en memoria RAM, disco duro y procesador es alta.
  - El fallo del servidor conlleva al fallo general de todos los sistemas.

Figura 31. **Diagrama de despliegue con un servidor**



Fuente: elaboración propia con MS Visio 2007.

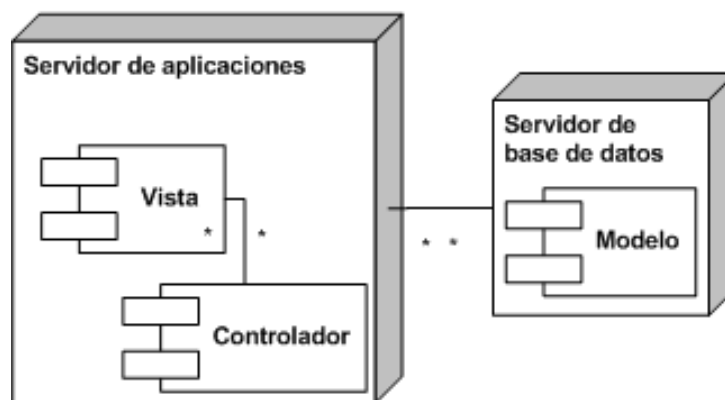
### 6.5.2. Dos servidores

Es la configuración más comúnmente usada al implementar el patrón MVC. La capa de datos tiene su servidor dedicado, mientras que el servidor de aplicaciones se encarga del controlador y la vista, estos se comunican mediante TCP/IP para la transmisión de datos.

- Ventajas
  - La carga de trabajo se reparte entre los dos servidores.
  - En la base de datos en la capa de modelo puede implementarse la alta disponibilidad, con configuraciones como cluster, replicación, entre otros.

- Se evitan conflictos de configuración de los servidores, ya que cada uno tiene una tarea específica, estos tienen su propia configuración.
- El acoplamiento se reduce, si se quiere cambiar de motor de base de datos, la instalación, configuración y despliegue es más fácil.
- El sistema es escalable ya que permite agregar servidores en las capas de acuerdo a la demanda de transacciones.
- Desventajas
  - Se complica la sincronización entre versiones de componentes.
  - La depuración de errores es más difícil.
  - Se aumenta el esfuerzo al administrar servidores.
  - La configuración errónea de políticas de seguridad puede provocar problemas de comunicación entre los servidores.
  - Problemas en los enlaces de comunicación entre servidores, puede hacer que los sistemas no funcionen si no cuentan con mecanismos de alta disponibilidad.

Figura 32. **Diagrama de despliegue con dos servidores**



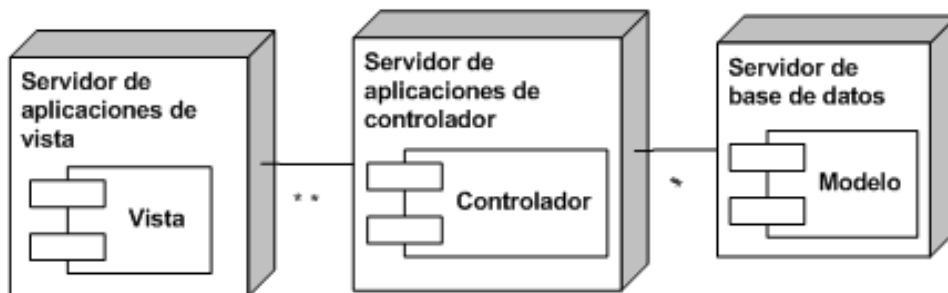
Fuente: elaboración propia con MS Visio 2007.

### 6.5.3. Tres o más servidores

Configuración donde cada capa está en un servidor distinto, es decir, un servidor dedicado para el modelo, uno para el controlador y uno último para la vista. Como la comunicación entre el controlador y la vista es en base a servicios, es posible crear adaptadores para que la vista pueda ser escrita en otro lenguaje, por ejemplo .NET con el API MVC y pueda comunicarse con el controlador, en este caso, escrito en PHP.

- Ventajas
  - Todas las ventajas de la configuración de dos servidores.
  - Mediante adaptadores se puede implementar la vista en otro lenguaje y aumentar la portabilidad.
  - La vista puede separarse en varios servidores, depende de los requerimientos.
- Desventajas
  - Todas las desventajas de la configuración de dos servidores.

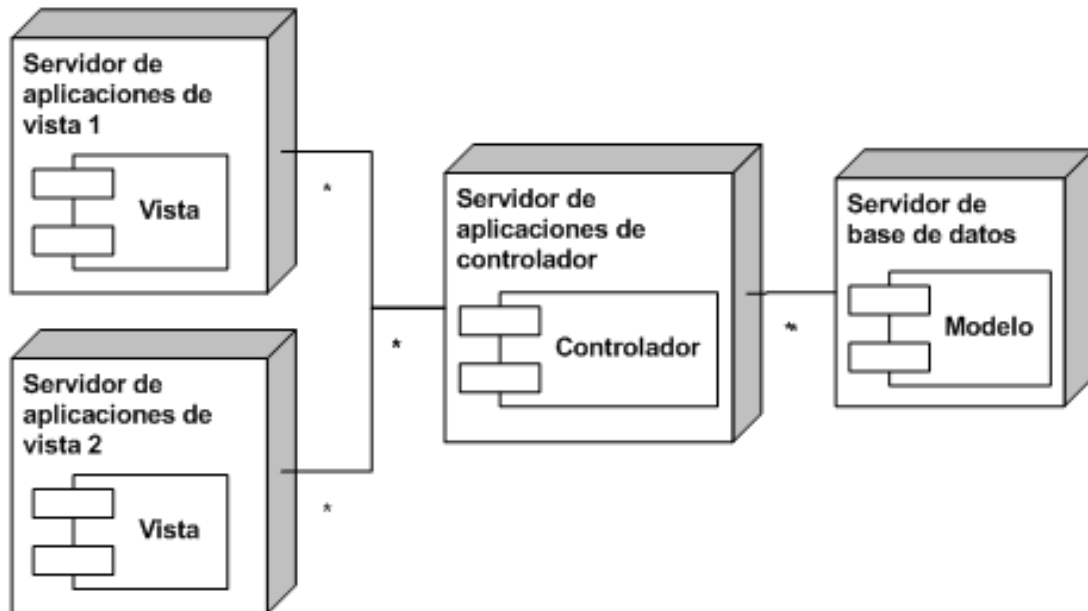
Figura 33. Diagrama de despliegue 3 servidores



Fuente: elaboración propia con MS Visio 2007.



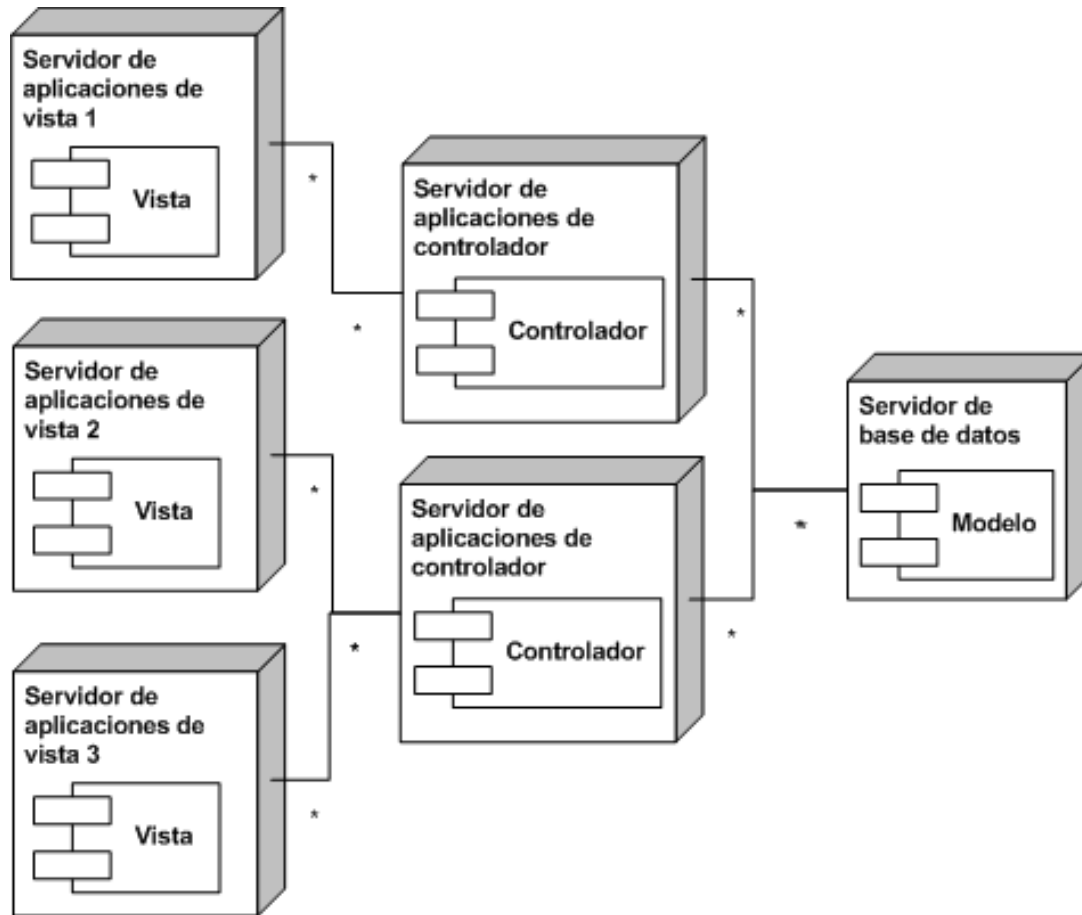
Figura 34. Diagrama de despliegue 3 o más servidores



Fuente: elaboración propia con MS Visio 2007.

Tanto el servidor de la vista como del controlador pueden ser desplegados en varios servidores depende del requerimiento, produciendo aplicaciones distribuidas, escalables, modificables, portables e integrables.

Figura 35. Diagrama de despliegue de aplicaciones distribuidas

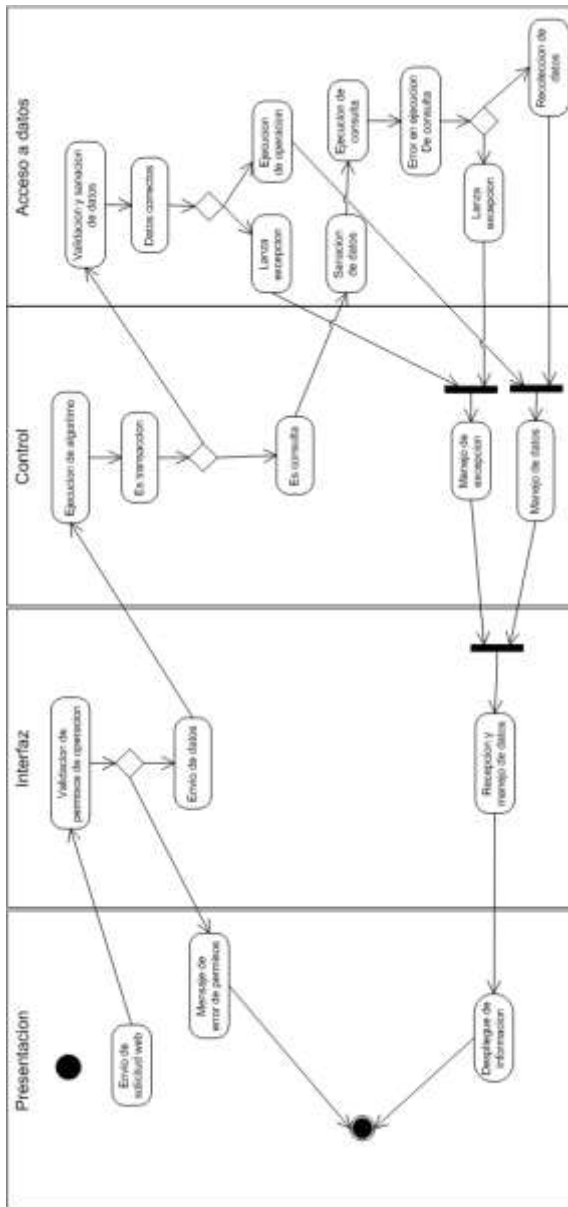


Fuente: elaboración propia con MS Visio 2007.

## 6.6. Interacción entre capas

Las capas se comunican de distintos modos, ya sea por instancia o por adaptadores como se mencionó en los capítulos anteriores. A continuación se presenta un diagrama de actividades del proceso de comunicación de forma resumida.

Figura 36. **Proceso de comunicación entre capas**



Fuente: elaboración propia con MS Visio 2007.



## **7. IMPLEMENTACIÓN EN LA INSTITUCIÓN DEL REGISTRO GENERAL DE LA PROPIEDAD**

El siguiente capítulo describe la institución del Registro General de la Propiedad y la experiencia de implementación de la arquitectura SKRN.

### **7.1. Registro General de la Propiedad**

Institución pública, encargada de inscripciones, anotaciones, cancelaciones de los actos y contratos relativos al dominio, además de derechos reales sobre bienes inmuebles o muebles identificables. Según la Constitución de la República de Guatemala, deberá ser organizado a efecto de que cada departamento o región, que la ley específica determine, se establezca su propio registro de la propiedad y el respectivo catastro fiscal.

El Registro General de la Propiedad cuenta con la dirección de informática, encargada de la creación de aplicaciones para el manejo de la información de la institución. Las aplicaciones se han clasificado en:

- Administrativas
- Registrales
- Informáticas
- Financieras

## 7.2. Implementación

En esta sección se describirá como se implementó, integro y desplegó la arquitectura en las aplicaciones de la institución. La siguiente tabla contabiliza las 24 aplicaciones que contiene la institución.

Tabla II. **Contabilización de aplicaciones y módulos del RGP**

<b>Descripción</b>	<b>Porcentaje</b>
Aplicaciones del departamento de informática que usan la arquitectura SKRN	34 %
Aplicaciones del departamento de informática que NO usan la arquitectura SKRN	58 %
Aplicaciones de terceros	8 %

Fuente: elaboración propia con MS Word 2007.

### 7.2.1. Integración con aplicaciones anteriores y de terceros

Anteriormente la dirección de informática tenía aplicaciones creadas en lenguaje PHP, para las nuevas aplicaciones a implementar se decidió crearlas sobre una arquitectura para estandarizar la codificación de estas. Además de la dirección de informática, el RGP tiene subcontratado otro departamento de informática, cuyas tecnologías utilizan Java como lenguaje de programación. Estas aplicaciones deben tener comunicación con las de la dirección, así que se implementan servicios web para lograr este objetivo.

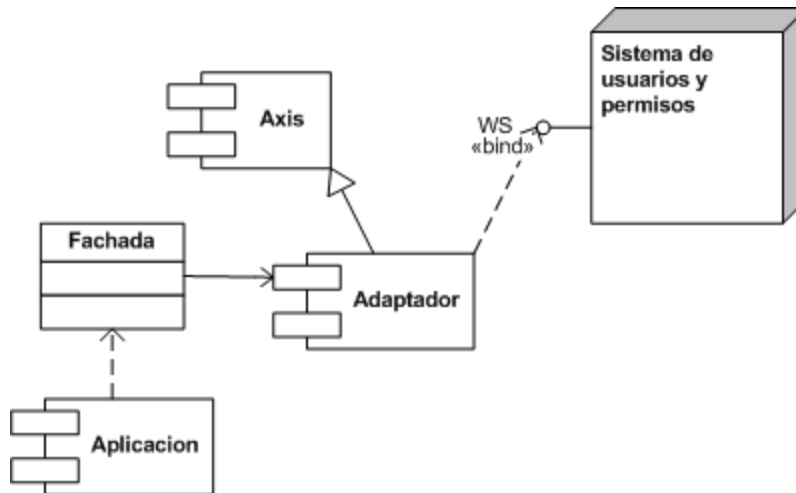
Este requerimiento de comunicación entre aplicaciones desarrolladas en diferentes tecnologías, se alcanzó a través de la arquitectura SKRN. Como se mencionó en el capítulo 5, esta arquitectura brinda de comunicación de la capa de interfaz con la de control a través de servicios web, permitiendo comunicación con aplicaciones de terceros de esa forma.

Algunas aplicaciones deben comunicarse con hardware, como lectores de huella digital, impresoras, escáneres, entre otras, esto no es posible con aplicaciones web, así que deben ser desarrolladas como aplicaciones de escritorio. Estas aplicaciones deben tener posibilidad que los usuarios usen los permisos que se manejan en la arquitectura, y ya que es basado en servicios, esta integración es posible.

En la institución se utiliza el lenguaje Java (de preferencia) para el desarrollo de aplicaciones de escritorio, se utiliza la librería axis y axis2 provistas por apache.org para la comunicación entre aplicaciones de escritorio y aplicaciones web.

Este componente crea una serie de clases a modo adaptador, que comunica Java con el servicio web del sistema de roles y servicios, Estas clases se implementan en una clase fachada, que realiza la implementación de las clases de adaptación, de este modo la aplicación que requiere esta comunicación solo realiza las llamadas mediante la clase de fachada.

Figura 37. **Diagrama de aplicación de escritorio**



Fuente: elaboración propia con MS Visio 2007.

### 7.2.2. Seguridad

Las aplicaciones se utilizan dentro de la intranet de la institución, no es posible que personas externas desde internet, accedan a los datos, estos son exclusivos para uso interno. Con esta condición no se implementaron certificados de encriptación como SSL o TLS, aunque en un futuro puede ser así.

Los sistemas nuevos implementan los permisos y autenticación del sistema de usuarios y permisos, así que una persona debe tener sus credenciales para realizar modificaciones sobre los datos, algunas de las aplicaciones antiguas y de escritorio utilizan estos mismos métodos de autenticación y operación, así que una persona sin los roles correctos no puede modificar información, y si la bitácora está activada y el sistema implementa el acceso a datos, cualquier registro será registrado por el sistema.



La institución utiliza un Active Directory para el control de usuarios, pero este no se enlaza con el sistema de usuarios y permisos, pues no hay una interfaz que una estos dos sistemas y administrativamente no se requiere. En los sistemas anteriores no se implementó el sistema de permisos, ya contaban con su propio gestor de usuarios, solo sistemas con codificación tradicional que se crearon concurrentemente a la implementación de la arquitectura, se adaptaron al sistema de roles y permisos.

### **7.3. Métricas de mejora de proceso**

Gracias a la implementación de la arquitectura, varios procesos han mejorado desde entonces, entre ellos tenemos:

- **Requerimiento de usuarios:** ya que los usuarios y permisos de todos los sistemas están integrados en un solo lugar, la creación y mantenimiento de este se facilita, antes se tenía que verificar si un usuario ya se existía en todos los sistemas, este proceso se reduce en un 33 % del tiempo original.
- **Recuperación de credenciales:** ya que todos los sistemas utilizan una sola credencial, cuando se pierde u olvida, se tiene un mecanismo de recuperación, ha disminuido las llamadas a servicio técnico para este requerimiento en un 80 %.
- **Requerimiento de roles para un sistema:** los roles están contenidos en el sistema de usuarios y permisos, y se utilizan interfaces para dicha configuración, no es necesario modificar código fuente pues estos permisos están parametrizados, la optimización de creación de nuevos permisos de acceso ha incrementado en un 100 %.

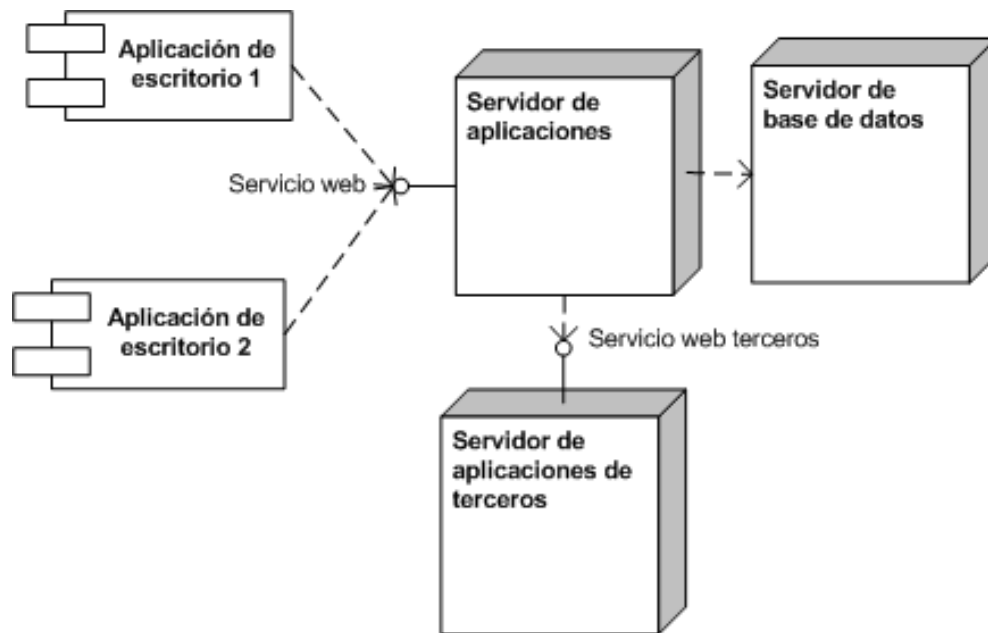
- Cambios de la GUI: el sistema de usuarios y permisos también contiene un listado de scripts para la visualización asociados a los sistemas. si se requiere un distinto color, tipo de letra, entre otros, estos cambios están totalmente parametrizados y no se tiene que modificar código fuente además, los cambios son instantáneos.
- Cambios de algoritmos: cuando existe un requerimiento nuevo o actualización de uno ya existente se tenía que comprender y trazar el funcionamiento de una página, con la estandarización del flujo del programa este esfuerzo se reduce a un 30 % del tiempo original.
- Pruebas de algoritmos: los algoritmos y consultas están encapsuladas en objetos, esto permite realizar pruebas unitarias tanto de archivos de control como de consultas de forma directa al solo instanciar el objeto, el esfuerzo de probar código se reduce en un 75 %.
- Generación y acceso a documentación: todas las páginas tienen el widget de documentación, está disponible para que el usuario obtenga detalles de cómo funciona la página que está visualizando, para rellenar de información, solo se requiere usar el módulo de documentación en el sistema de usuarios y permisos. Esta misma información puede ser generada en un documento PDF, pero por motivos administrativos se debe requerir al departamento de informática por ser un documento interno de documentación, el tiempo de despliegue de documentación se reduce a un 20 % del tiempo original.

### 7.3.1. Despliegue de la arquitectura

Se utilizó la configuración de dos servidores, se tiene un servidor dedicado para el modelo (la base de datos) y otro exclusivo para la vista y el controlador.

Las aplicaciones de terceros tienen su propio servidor y se comunican con las aplicaciones de la dirección de informática por servicios web. Las aplicaciones de escritorio de la dirección de informática también se integran de ese modo con las aplicaciones web.

Figura 38. Diagrama de despliegue de servidores



Fuente: elaboración propia con MS Visio 2007.

#### 7.4. Experiencia de implementación

A continuación se listan las experiencias en la implementación de la arquitectura SKRN.

- Problemas de implementación: la utilización de una arquitectura a comparación de una codificación tradicional en PHP es un cambio muy drástico, los programadores pueden rechazar su utilización por miedo al cambio.
- Requerimientos que afectaron la arquitectura: entre los requerimientos que hicieron evolucionar la arquitectura está:
  - Versatilidad de la GUI
  - La generación de documentación automática asociada a roles
  - Bitácora de manipulación de datos.
  - Utilización de metadatos de la base de datos para la validación de datos en capa e acceso a datos.
- Tiempo de implementación: desde el inicio de la arquitectura, hasta su fase más madura, transcurrieron 6 meses, implementándose en ese tiempo en dos sistemas.
- Sistemas involucrados: originalmente el sistema de usuarios y permisos era solo una implementación de la arquitectura, pero luego se consideró que además de ser así, debía ser parte de la arquitectura misma, por todas las funcionalidades y facilidades genéricas que ofrece hacia esta.
- Personas que participaron: solamente una persona.

- Problemas de implementación con los usuarios: no, la arquitectura es totalmente transparente al usuario.
- Problemas de implementación con los desarrolladores: la idea es que reutilicen los componentes que conforman la arquitectura, a cambio, se copian y pegan los componentes o parte de estos en sus propios proyectos.
- Versionamiento de componentes: cada componente puede actuar independiente, por la naturaleza del patrón MVC. Cada componente está escrito de modo que tenga compatibilidad hacia atrás, pero esto depende de la rama principal de su respectiva versión, una versión de la rama 1.0 va a ser compatible con todas hacia atrás, una de la versión de la rama 2.0 no será compatible con las del 1.0, pero si con todas las de su propia rama.



## **8. EVALUACIÓN DE ARQUITECTURA**

El siguiente capítulo describe los pasos de la evaluación de la arquitectura anteriormente descrita, para determinar que los atributos de calidad del sistema son los correctos y así prevenir problemas que en un futuro puedan suceder, una evaluación temprana es un mecanismo relativamente barato para evitar desastres futuros.

### **8.1. Presentación**

En esta etapa se explica el proceso de la evaluación ATAM, se presentan los 9 pasos que lo conforman, se explican las técnicas a utilizarse para la generación de ideas, priorización de escenarios, etc, explicándose los productos a obtenerse al final de la evaluación.

#### **8.1.1. Presentación de ATAM**

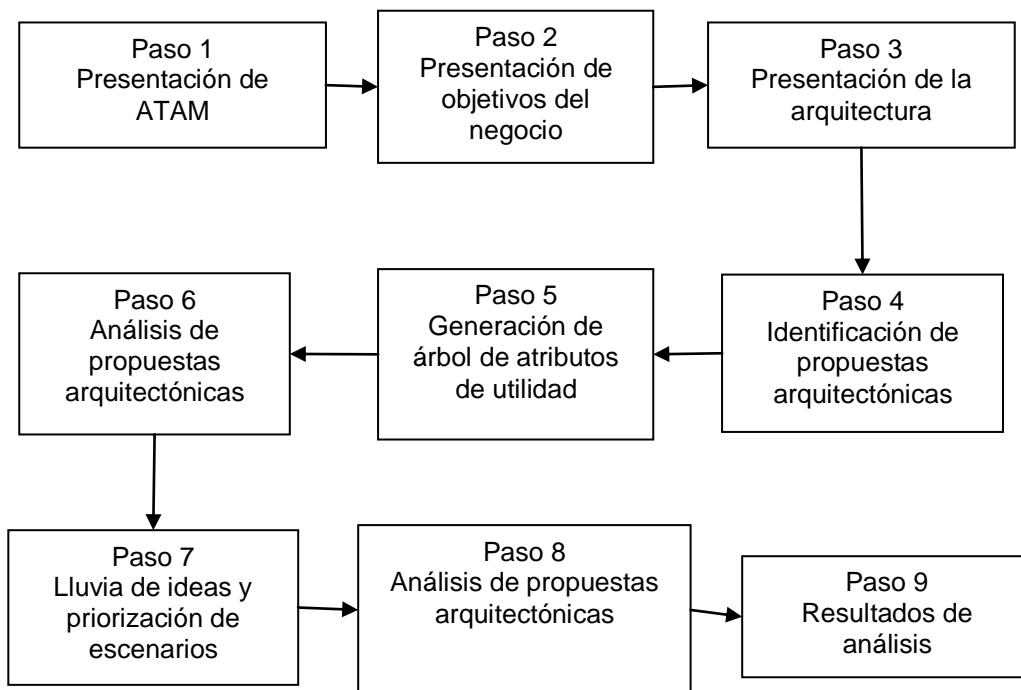
Es un método de evaluación de arquitectura de software desarrollado e impulsado por el Instituto de Ingeniería de Software, (Software Engineering Institute, SEI), este centra su actividad de evaluación en la interacción entre los diferentes atributos de calidad arquitectónica y basa sus evaluaciones sobre escenarios desarrollados por los involucrados.

El propósito de ATAM es evaluar las consecuencias de una decisión de arquitectura en función de las necesidades de los atributos de calidad. Los mayores objetos de ATAM son:

- Obtener y refinar una declaración precisa de las necesidades de los atributos de calidad en función de la arquitectura.
- Obtener y refinar una exposición precisa de las decisiones del diseño arquitectónico.
- Evaluar las decisiones de diseño arquitectónico para determinar si se refieren a los requisitos de los atributos de calidad.

Son 9 pasos los involucrados en esta evaluación, incluido este:

Figura 39. **Diagrama de pasos del proceso de análisis ATAM**



Fuente: elaboración propia con MS Word 2007.



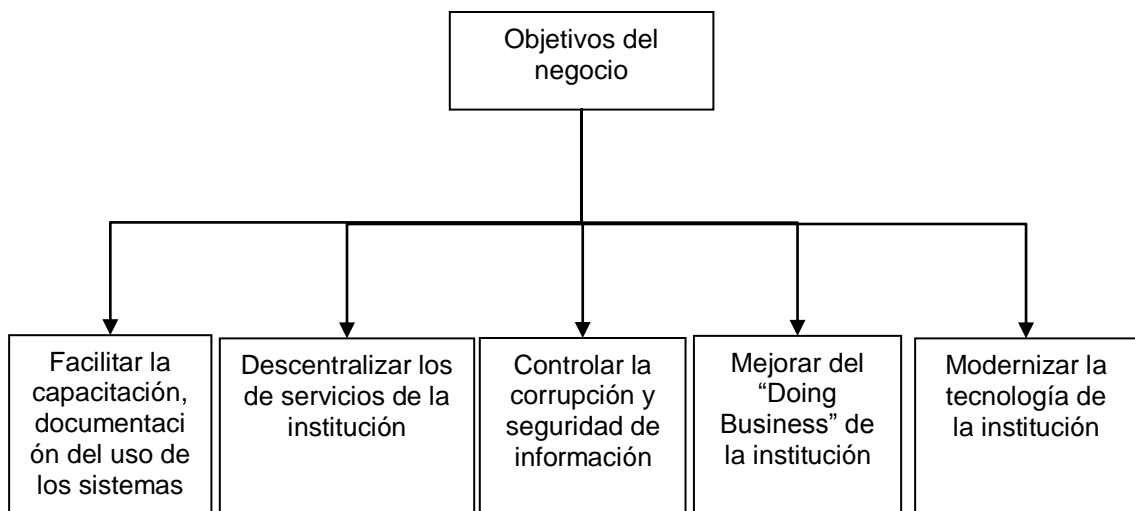
Los resultados que se obtendrán son los riesgos descubiertos, posibles problemas de arquitectura, posibles conflictos, puntos de sensibilidad y propiedades de estos.

### 8.1.2. Presentación de objetivos del negocio

Se describen que objetivos del negocio son los motivadores del esfuerzo del desarrollo y por lo tanto cuales deberían ser las principales finalidades arquitectónicas.

Se requieren aplicaciones intuitivas, fáciles de usar y estandarizadas en su interfaz y desarrollo; compatibilidad hacia atrás, e integración con otros sistemas antiguos o de terceros. La siguiente gráfica agrupa los principales objetivos del negocio.

Figura 40. **Objetivos del negocio**



Fuente: elaboración propia con MS Word 2007.

### **8.1.2.1. Facilitar la capacitación, documentación del uso de los sistemas**

Las personas involucradas son los usuarios de la institución, las aplicaciones estarán enfocadas a distintos usuarios del RGP, cada uno con sus requerimientos y demandas. Algunas aplicaciones son específicas a algún departamento, y otras tendrán roles que involucren muchos departamentos de la institución.

Los usuarios generalmente son personas mayores, y su interacción con la informática está limitada por sus antiguos paradigmas de trabajo, esto requiere que los sistemas sean cómodos a la vista y simples en interfaz, que cada página tenga tareas específicas y bien definidas.

Las aplicaciones deben tener acceso a la documentación, las personas se agobian al no entender cómo funciona una aplicación por más simple que sea, además, estos demandan de capacitaciones y manuales. Algunos sistemas tendrán gran cantidad de roles, por lo tanto, la generación de manuales se vuelve costosa en tiempo, he ahí el porqué de la generación automática para esa gran cantidad de información y que sea consistente con la ayuda en línea que provee la arquitectura.

### **8.1.2.2. Descentralizar de servicios de la institución**

Para cumplir con el Artículo 130 de la Constitución Política de la República de Guatemala y brindar un servicio eficiente y eficaz a los usuarios en todo el país, se debe descentralizar las funciones de la institución.

La institución tiene dos departamentos de informática, uno interno y otro de terceros, estos dos crean las aplicaciones para los procesos registrales, administrativos, financieros e informáticos, pero cada uno utiliza servidores en distintas ubicaciones, distintos manejadores de bases de datos y lenguajes de programación.

Además de eso, el departamento de informática tiene sistemas antiguos y sistemas que se crearon conjuntamente con la implementación de la arquitectura. La información de cada sistema está relacionada, por ejemplo, existe un sistema antiguo de recursos humanos, y lleva la información de los empleados en el, estos datos deben ser compartidos con sistemas nuevos, por lo tanto se requería una interfaz que comunicará los sistemas nuevos con este viejo.

Otro ejemplo es el sistema de recepción de documentos en atención al cliente, debe comunicarse con el sistema de certificaciones de propiedades, por lo tanto se realizó una interfaz mediante servicios para realizar dicha tarea.

### **8.1.2.3. Controlar la corrupción y seguridad de la información**

La seguridad es un tema que provee de complejidad extra en las aplicaciones, pero es necesaria para mantener la integridad, confidencialidad, entre otras.

Un rol no puede entrar a cierta parte de la aplicación, o bien, puede entrar, pero no modificar la información contenida en el, las aplicaciones deben proveer de mecanismos de acceso hacia las páginas y el accionamiento de las operaciones dentro de las mismas.

Además, debe proveer protección de los datos, ataques como inyección SQL pueden ser peligrosos para la integridad de la información. La configuración de usuarios de base de datos correcta para incrementar esta protección. La validación de los datos es importante para evitar inconsistencias de estos y los mensajes de estos errores deben ser claros y deben ser provistos al usuario cuando sucedan.

El tema de corrupción es crítico, se debe proveer de mecanismos que tracen y registren la actividad de las personas que manipulan los datos, con el fin de evitar corrupción de información.

Así mismo, proveer a la auditoría, herramientas para la visualización de cambios de datos, informes y reportes de los procesos, evaluación de desempeño, entre otras.

#### **8.1.2.4. Mejorar del “Doing Bussiness” de la institución**

Los procesos registrales son evaluados por instituciones como el Banco Mundial, este investiga las facilidades de inversión de los países miembros y analiza la innovación de los procesos, el tiempo y costos de estos.

Las herramientas deben ser flexibles y fáciles, para crear o modificar estos reportes eficaz y eficientemente. El mejoramiento de los procesos mediante la informática, permitirá a la institución obtener los mejores lugares en las instancias registrales públicas de propiedad de Latinoamérica.

#### **8.1.2.5. Modernizar la tecnología de la institución**

Con el tiempo, los sistemas evolucionan adquiriendo nuevas funcionalidades que son requeridas por los usuarios. Este objetivo del negocio se enfoca en la parte informática de la institución, tanto del software como la actualización del hardware, trazar el código para localizar los errores es difícil si este es muy complejo y si todo está en un solo archivo, la arquitectura, debe estar separada en capas y subcapas, obteniendo así, archivos no tan complejos.

La reducción de complejidad y especialización de código reduce el esfuerzo de pruebas, búsqueda de errores, además la escritura de código en objetos provee de la realización de pruebas unitarias de código.

El código debe ser modular, si es cambiado en una capa o subcapa, no debe interferir con el funcionamiento de otra, y solo se deberían cambiar pocos archivos al actualizar la aplicación.

#### **8.1.3. Presentación de arquitectura**

La arquitectura a evaluar ha sido presentada en los capítulos previos de este trabajo, para visualizar todos los detalles, ver capítulos del 3 al 6.

### **8.2. Investigación y análisis**

Se identificarán las propuestas arquitectónicas y se relacionarán al árbol de utilidad, este los priorizará respecto los objetivos del negocio y se abstraerán los escenarios más importantes para posteriormente, ser analizados.

### **8.2.1. Identificación de propuestas arquitectónicas**

Son los enfoques arquitectónicos identificados por el arquitecto de software, pero no son analizados aun.

- Utilización de acceso a datos para transacciones y consultas
- Uso del patrón vista modelo controlador
- Comunicación de la vista con el controlador por servicio web
- Centralización de usuarios, sistemas, roles y permisos
- Centralización de estilos CSS para las vistas de la aplicación
- Utilización de adaptador para comunicar la vista con el controlador
- Implementación de bitácoras de ingresos y manipulación de datos
- Servicio web de autenticación de ingresos y permisos
- Servicios web de comunicación entre sistemas internos y externos

### **8.2.2. Generación de árbol de atributos de utilidad**

Un atributo de calidad provee de un mecanismo jerárquico para trasladar directa y eficientemente los objetivos del negocio de un sistema a escenarios concretos para probar los atributos de calidad.

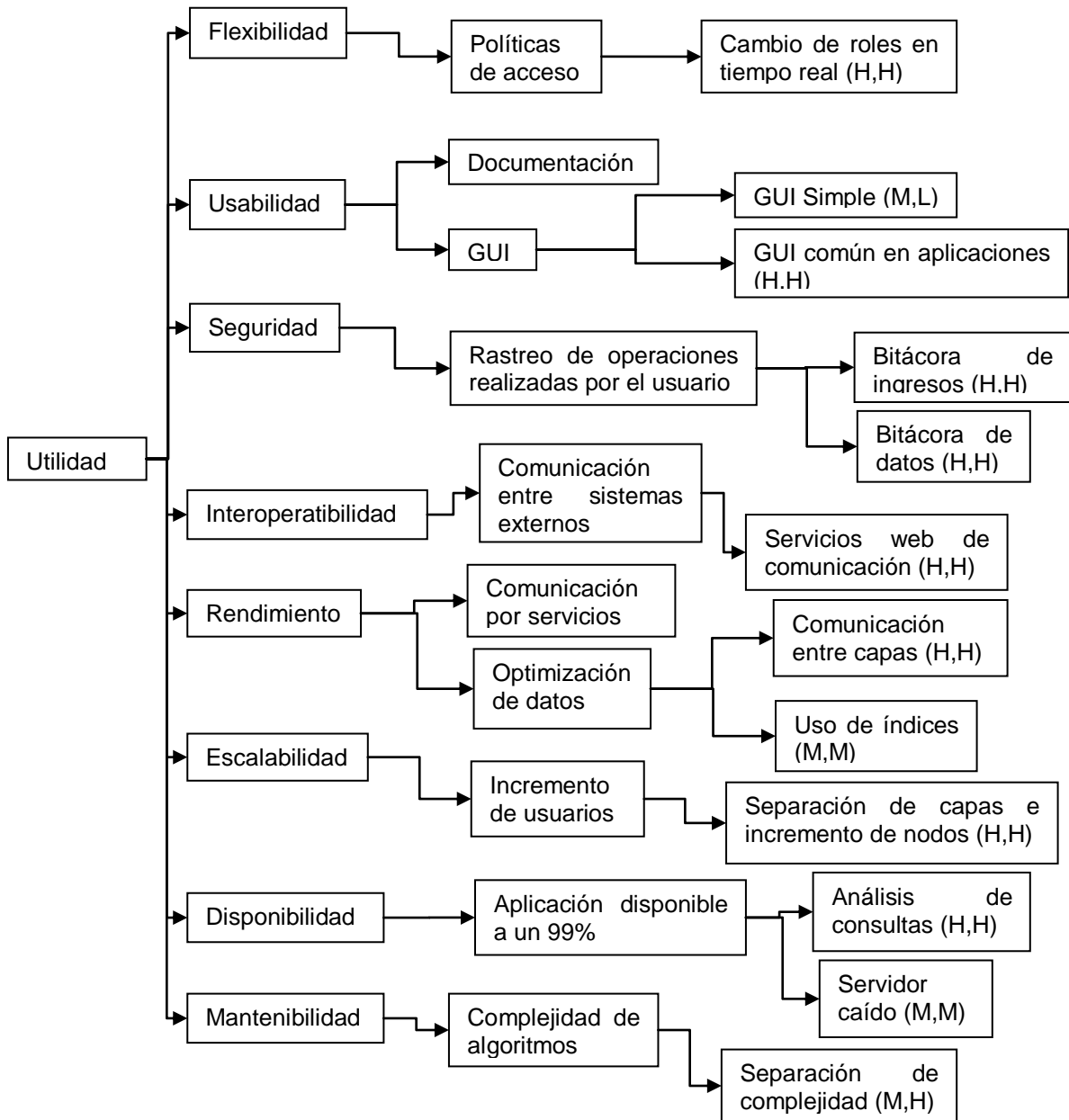
Los atributos de calidad que componen un sistema, tales como rendimiento, disponibilidad, seguridad, entre otros, se especifican junto a escenarios que los afecten, clasificándose según sus estímulos y respuestas para ser al final, priorizados.

A continuación se muestran los atributos de calidad más importantes para la institución según sus objetivos del negocio, se presentan sus propiedades y los posibles escenarios en los cuales se implementan. La clasificación tiene la siguiente anotación:

- H High (prioridad alta)
- M Medium (prioridad media)
- L Low (prioridad baja)

La priorización del árbol de utilidad está formado por dos dimensiones, por la importancia de cada nodo para el éxito del sistema y el grado de percepción del riesgo que plantea el logro de este nodo, en otras palabras, la facilidad de la arquitectura en lograr los atributos de calidad.

Figura 41. Diagrama de árbol de utilidad



Fuente: elaboración propia con MS Word 2007.



Tabla III. **Atributos de calidad requeridos en la institución**

<b>Atributo de calidad</b>	<b>Flexibilidad</b>
Propiedad del atributo	Políticas de acceso.
Escenario	1. Cuando se realizan cambios en los roles de usuarios, deben aplicarse en tiempo real (H, H).
Escenario	2. Cambios en la aplicación deben ser inmediatos en el ambiente de producción (H, M).
Propiedad del atributo	No se debe modificar el modelo de datos.
Escenario	3. Se requiere una categoría nueva (L, L).
<b>Atributo de calidad</b>	<b>Usabilidad</b>
Propiedad del atributo	Acceso a la documentación.
Escenario	4. El usuario debe tener documentación de la aplicación que usará (L, L).
Escenario	5. El usuario no tiene a la mano un manual impreso o electrónico, pero debe saber cómo usar una vista de la aplicación (M, L).
Propiedad del atributo	Conocimiento de las acciones al utilizar el sistema.
Escenario	6. Al operar información, el sistema debe mandar mensaje del estado de alguna operación realizada (M, M).
Propiedad del atributo	Manipulación de datos.
Escenario	7. La información no debe manipularse directamente en la base de datos, siempre debe existir una GUI (M, M).
Propiedad del atributo	GUI consistente.
Escenario	8. La GUI debe ser común en las aplicaciones para una adaptación rápida (H,H).
Escenario	9. Las vistas al ser simples, no tienen muchas operaciones y su adaptación a esta deberá ser rápida (H, M).
Propiedad del atributo	Manipulación de credenciales.
Escenario	10. Debe existir mecanismo de recuperación de credenciales (H, M).
Escenario	11. Un usuario tiene información de a que sistemas puede acceder (H, H).

Continuación de la tabla III.

<b>Atributo de calidad</b>	<b>Seguridad</b>
Propiedad del atributo	Rastreo de operaciones realizadas por el usuario.
Escenario	12. Registro de cambios realizados en la base de datos (H, H).
Escenario	13. Registro de ingresos al sistema (H, H).
Propiedad del atributo	Confidencialidad de información.
Escenario	14. El usuario debe acceder solo a información que le pertenece (H, H).
Propiedad del atributo	Los usuarios están centralizados en un solo lugar.
Escenario	15. Un usuario solo necesita un nombre de usuario y contraseña para acceder a los sistemas (M, M).
Propiedad del atributo	Consistencia de los datos.
Escenario	16. Los datos no pueden tener valores fuera de su dominio (M, L).
Escenario	17. Debe existir integridad referencial (M, M).
<b>Atributo de calidad</b>	<b>Interoperabilidad</b>
Propiedad del atributo	Comunicación entre sistemas internos, externos y de terceros.
Escenario	18. Existe una aplicación de terceros con el cual hay que compartir información (H, M).
Escenario	19. Existen aplicaciones antiguas en las cuales se debe compartir información (M, M).
Escenario	20. Existe aplicación de escritorio que no debe comunicarse directamente con la base de datos al manipular información (H, M).
Escenario	21. Aplicaciones de escritorio o de terceros requieren conectarse a los sistemas de la arquitectura (H, H).
<b>Atributo de calidad</b>	<b>Adaptabilidad</b>
Propiedad del atributo	Acomodación de nuevos requerimientos.
Escenario	22. Cambios en una tabla de la base de datos, no afecta cuando se debe operar en la GUI, a menos que sea un campo obligatorio (H, M).
Escenario	23. Se requiere cambio de un algoritmo pero mantener el antiguo (M, M).
Escenario	24. Se requiere un Look and Feel Nuevo, este debe implementarse en todos los sistemas (M, M).

Continuación de la tabla III.

<b>Atributo de calidad</b>	<b>Mantenibilidad</b>
Propiedad del atributo	Pruebas de algoritmos.
Escenario	25. Se requiere probar el funcionamiento de un algoritmo, se realiza una prueba unitaria, ya sea en el control o en la consulta (H, H).
Escenario	26. Se requiere un algoritmo complejo, debe separarse en varias funciones en las distintas capas si lo requiere para funcionar (H, H).
Propiedad del atributo	Estructuración de sistemas.
Escenario	27. Se crea una vista nueva, solo se debe registrar en el sistema de usuarios y permisos, asignar los permisos a los roles y esta estar disponible (H, M).
Escenario	28. Vistas que ya no se usan, no deben eliminarse, solo marcar como desactualizadas (L, L).
Propiedad del atributo	Acceso a la base de datos.
Escenario	29. La manipulación de datos debe ser transparente (H,H).
Escenario	30. Las consultas deben facilitar el acceso a la visualización de la información (H,M).
<b>Atributo de calidad</b>	<b>Escalabilidad</b>
Propiedad del atributo	Incremento del número de usuarios.
Escenario	33. La carga de trabajo de un servidor es demasiado alta y afecta el rendimiento (H, H).
Escenario	34. La aplicación no está disponible o tiene un rendimiento por debajo de lo esperado.
<b>Atributo de calidad</b>	<b>Rendimiento</b>
Propiedad del atributo	La comunicación entre sistemas es por servicios.
Escenario	35. La comunicación entre la capa de controlador y la capa de vista puede ser por servicios (L,L).
Escenario	36. La comunicación entre el servidor y las aplicaciones de escritorio es por servicios (M,M).
Propiedad del atributo	Buenas prácticas de optimización.
Escenario	37. Utilización de llaves primarias y foráneas (H,M).

Continuación de la tabla III.

Escenario	38. Capacidad de cambiar la configuración de comunicación entre las capas (H,H).
Propiedad del atributo	Incremento del número de usuarios.
Escenario	39. Aumenta el número de transacciones y consultas que se ejecutan en las aplicaciones (H, M).
<b>Atributo de calidad</b>	<b>Disponibilidad</b>
Propiedad del atributo	La aplicación está en línea el 99 % del tiempo.
Escenario	40. Uno de los servidores se cae y no está disponible para proveer servicios (M,M).
Escenario	41. Una consulta bloqueo alguna tabla y no deja realizar consultas a demás usuarios (M,M).

Fuente: elaboración propia con MS Word 2007.

Con base en experiencia en la institución y a las opiniones obtenidas de los usuarios de los sistemas que utilizan la arquitectura, a través de entrevistas realizadas, se obtuvieron los siguientes escenarios como los principales a analizar.

Tabla IV. **Tabla de escenarios de mayor relevancia**

<b>Escenario #</b>	<b>Estímulo</b>	<b>Ambiente</b>	<b>Respuesta</b>
1	Se cambian los permisos de un usuario.	Producción	Usuarios en tiempo real no pueden acceder a las páginas con nuevos roles, aun con la sesión abierta.
11	El usuario quiere saber qué sistemas y opciones tiene disponibles.	Producción	Se muestra un listado de sistemas que tiene disponibles.

Continuación de la tabla IV.

12	Un usuario intenta persuadir que no realizó cierto cambio en la base de datos.	Producción	Se revisa la bitácora con base en la fecha y el usuario.
13	Un usuario intenta persuadir que no ingresó al sistema cierto día.	Producción	Se revisa la bitácora de ingresos.
39	La cantidad de usuarios incrementa.	Producción	Se incrementan los recursos del servidor.
21	Se requieren algoritmos para la ejecución de tareas y consulta de datos.	Producción	Utilización de servicios web para encapsular operaciones y conexiones a base de datos.
25	Un algoritmo no funciona como debe, y hay que verificarlo.	Desarrollo	Se realiza una instancia del objeto en un archivo de prueba y se envían los parámetros necesarios para su funcionamiento.
26	Un algoritmo utiliza muchos datos y operaciones.	Desarrollo	Se separa en operaciones básicas y se integra en un método.
41	Una consulta o transacción bloquea otras operaciones en curso.	Producción	Se extrae el plan de ejecución de la consulta, se analiza y se reescribe para optimizarla.
33	Un servidor requiere demasiados recursos y requisiciones, se vuelve ineficiente.	Producción	Se analiza que capa sobrecarga el servidor y se separa para dividir tareas.
8	Se requiere un cambio en la GUI para una vista más cómoda.	Producción	Se modifica el estilo que las páginas implementan y actualiza todos los sistemas que la usan.

Fuente: elaboración propia con MS Word 2007.

### 8.2.3. Análisis de propuestas arquitectónicas

Los escenarios elegidos, contrastados con las propuestas arquitectónicas, muestran las ventajas de la arquitectura actual sobre otras alternativas, también se identifican riesgos que suceden o pudieran suceder. De dichos riesgos se obtendrán mejoras para la arquitectura en análisis de la siguiente sección.

Tabla V. **Centralización de usuarios, sistemas, roles y permisos**

<b>Atributo</b>	Flexibilidad.			
<b>Escenario</b>	Cuando se realizan cambios en los roles de usuarios, deben aplicarse en tiempo real.			
<b>Estímulo</b>	Se cambian los permisos de un usuario.			
<b>Respuesta</b>	El usuario en tiempo real no puede acceder a las páginas con los roles nuevos aun con la sesión abierta.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Módulo de roles en cada sistema	S1	T2	R1, R2	
Roles centralizados	S2	T1, T3	R3	R1, R2
<b>Razonamiento</b>	S1. Empeora la mantenibilidad S2. Empeora el rendimiento T1. Mejora la mantenibilidad, aumenta la confiabilidad T2. Mejora el rendimiento, disminuye la confiabilidad T3. Mejora la reusabilidad R1. Aumenta la dependencia entre la aplicación y la BD R2. Aumento de inconsistencias de datos R3. No disponibilidad de la información centralizada			

Fuente: elaboración propia con MS Word 2007.

Este escenario describe cuando a un usuario le son concedidos o revocados permisos para la manipulación de datos o acceso a opciones de los sistemas.

Cada sistema tiene un conjunto de roles con permisos asociados, estos son concedidos a los usuarios desde el sistema de usuarios y permisos donde se encuentra toda esa información. Cuando administrativamente se deben revocar o conceder permisos, estos cambios deben efectuarse en tiempo real.

Respecto la creación o modificación de un rol, ya que es parametrizada, estos cambios se efectúan en tiempo real, evitando así la modificación del código fuente.

- Puntos de sensibilidad (Sensitivity)

La verificación de los permisos de forma parametrizada provee de carga extra de trabajo al servidor, ya que por cada operación o permisos de visualización debe validar dichos permisos, reduciendo así, el rendimiento del sistema.

- Puntos de compensación (Tradeoff)

Al utilizar este mecanismo se aumenta la confiabilidad, ya que la información de estos permisos está centralizada y pueden ser consultados quienes tienen dichos roles o que roles tiene un usuario.

La mantenibilidad mejora, por la misma razón de estar centralizada la información. Modificaciones, creaciones, asignación de permisos, consenso y revocación de estos mismos.

Ambos elementos se logran impactando al rendimiento del sistema. Sin embargo, el impacto en el rendimiento no es significativo, además de acuerdo a los objetivos de la institución, es permisible dicho impacto.

- Riesgos

Si por algún motivo, el sistema de usuarios y permisos no se encuentra disponible, ni un usuario podrá acceder a cualquier sistema que dependa de este, hasta que funcione de nuevo.

- No riesgos

Si cada sistema tiene su propio gestor de usuarios y permisos, se dificulta el mantenimiento de estos y aumenta el riesgo de inconsistencias de los datos.

Tabla VI. **Uso del patrón MVC**

<b>Atributo</b>	Rendimiento.			
<b>Escenario</b>	Aumenta el número de transacciones y consultas que se ejecutan en las aplicaciones.			
<b>Estímulo</b>	La cantidad de usuarios incrementa.			
<b>Respuesta</b>	Se incrementan los recursos del servidor.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso del patrón MVC		T1, T2		
<b>Razonamiento</b>	T1 Aumenta escalabilidad T2 Aumenta rendimiento			

Fuente: elaboración propia con MS Word 2007.

Este escenario ejemplifica la situación que la cantidad de transacciones y consultas incrementan por unidad de tiempo.

Esto se debe a la cantidad incrementada de usuarios que usan el sistema y hace que el rendimiento disminuya. Mientras más usuarios, más



transacciones y llamadas a la base de datos habrá, por lo tanto los servidores necesitarán más recursos o la creación de nuevos nodos de servidores.

- Puntos de compensación (Tradeoff)

Al utilizar el patrón MVC, las tareas se especializan en cada capa de la aplicación, por el diseño de la arquitectura, se pueden mover aplicaciones enteras o una capa de una aplicación a nuevos nodos y aumentar el rendimiento. Además, la institución invierte en tecnología y el hardware utilizado es lo suficiente capaz como para soportar las exigencias de las aplicaciones que utilizan la arquitectura.

Tabla VII. **Implementación de bitácoras de manipulación de datos**

<b>Atributo</b>	Seguridad.			
<b>Escenario</b>	Registro de cambios realizados en la base de datos.			
<b>Estímulo</b>	Un usuario intenta persuadir que no realice cierto cambio en la base de datos.			
<b>Respuesta</b>	Se revisa la bitácora en base a la fecha y el usuario.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
No usar bitácoras	S2	T2 T3	R1	
Uso de bitácora	S1	T1	R2	R1
<b>Razonamiento</b>	S1. Disminuye rendimiento S2. Disminuye confiabilidad T1. Aumenta confiabilidad, disminuye rendimiento T2. Disminuye confiabilidad T3. Aumenta rendimiento R1. Usuarios pueden evadir responsabilidades R2. Alto tiempo de búsqueda en bitácoras			

Fuente: elaboración propia con MS Word 2007.

Este escenario ilustra, cuándo un usuario realiza una operación en la base de datos, por medio de una aplicación que implemente la arquitectura. Puede

llegar la situación que un usuario manipuló cierta información para malversarla o simplemente por confusión.

Se provee de un mecanismo de historial de los datos de la base de datos, almacenando quien, cuando y donde se manipuló información. De este modo los usuarios no pueden evadir auditorias que sean realizadas sobre las aplicaciones.

- Puntos de sensibilidad (Sensitivity)

La aplicación al tener la bitácora activada, requiere de recursos extra para el almacenamiento de esta información, esto conlleva a gastar espacio en disco, memoria y operaciones al realizar mantenimientos de datos.

- Puntos de compensación (Tradeoff)

Al tener un historial de datos, se aumenta la confiabilidad en la aplicación, de este modo los usuarios no pueden evadir responsabilidades al manipular la información de las aplicaciones.

- Riesgos

La utilización de bitácoras puede usar mucho mas información que los datos mismos, es importante llevar un control del espacio disponible para no saturar el disco y no dejar de almacenar datos nuevos.

- No riesgos

Los usuarios no evadirán responsabilidades de información, en una auditoria de sistemas, si alguna información es sospechosa, puede verificarse quien fue el autor de esa manipulación de datos.

Tabla VIII. **Implementación de bitácoras de ingresos**

<b>Atributo</b>	Seguridad.			
<b>Escenario</b>	Registro de ingresos al sistema.			
<b>Estímulo</b>	Un usuario intenta persuadir que no ingreso al sistema cierto día.			
<b>Respuesta</b>	Se revisa la bitácora en base a la fecha y el usuario.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
No usar bitácora	S1		R1	
Uso de bitácora		T1	R2	R1
<b>Razonamiento</b>	S1. Disminuye la confiabilidad T1. Aumenta la confiabilidad R1. Usuarios pueden evadir responsabilidades R2. Aumento de tamaño de espacio en bitácora			

Fuente: elaboración propia con MS Word 2007.

Este escenario muestra la situación, que un usuario entró a alguna aplicación para manipular datos, se puede saber quien, cuando y donde ingresó en el sistema, así como bloqueos de contraseña por intentar ingresar con credenciales incorrectas.

Puede llegar la situación que un usuario niega el ingreso a algún sistema, pero gracias a la bitácora de ingresos, se puede verificar que esto fue así. Algunos usuarios olvidan la contraseña de su credencial y al intentar ingresar, bloquean esta y culpan al sistema de que no los dejó entrar, la bitácora de

ingresos provee de información para corroborar las afirmaciones que los usuarios dan y así evadir responsabilidades.

- Puntos de compensación (Tradeoff)

Aumenta confiabilidad al registrar los ingresos de los usuarios, se puede verificar los intentos de ingreso, la hora y el lugar (mediante la dirección IP de donde se conectan) para validar sus afirmaciones.

- Riesgos

El aumento del tamaño de la bitácora podría saturar el espacio en disco, es importante verificar que estos datos no lo saturen.

- No riesgos

Los usuarios no podrán evadir responsabilidades al tenerse registrados los intentos de ingresos a los sistemas.

Tabla IX. **Servicio web de autenticación de ingresos y permisos**

<b>Atributo</b>	Interoperabilidad.			
<b>Entorno</b>	Aplicaciones de escritorio o de terceros requieren conectarse a los sistemas de la arquitectura.			
<b>Estímulo</b>	Se requieren algoritmos para la ejecución de tareas y consulta de datos.			
<b>Respuesta</b>	Utilización de servicios web para encapsular operaciones y conexiones a base de datos.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso de servicios web	S1	T1, T2, T4		R1
Conexión directa		T3	R1	

Continuación de la tabla IX.

<b>Razonamiento</b>	S1. Disminuye rendimiento. T1. Aumenta la integración con otros sistemas. T2. Aumenta la confiabilidad al encapsular conexiones y algoritmos. T3. Disminuye la seguridad al exponer conexiones y algoritmos en el código de aplicaciones de escritorio o de terceros. T4. Incrementa escalabilidad. R1. Conexiones a bases de datos pueden ser extraídas.
---------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fuente: elaboración propia con MS Word 2007.

Este escenario ilustra, cuando aplicaciones de terceros o de escritorio necesitan utilizar las características de la arquitectura, como lo es el sistema de usuarios y permisos para validar ingresos y datos.

Una aplicación de terceros puede ser modificada para que utilice las credenciales del sistema de usuarios y permisos, este requiere acceder a la información de usuarios y para esta realización, se cuenta con un servicio web con los métodos necesarios para el control de accesos a módulos, opciones e ingresos a dicha aplicación. Lo mismo sucede con las aplicaciones de escritorio que requieran accesos.

- Puntos de sensibilidad (Sensitivity)

El requerir consultas hacia los permisos de usuario, incrementa la carga de operaciones que debe realizar el Sistema de usuarios y permisos, disminuyendo el rendimiento de este al incrementarse el número de usuarios.

La utilización de servicios web también disminuye el rendimiento, ya que los datos deben ser enviados, viajar por la red e interpretados por los clientes.

- Puntos de compensación (Tradeoff)

El uso de servicios web incrementa la portabilidad e integración con otros sistemas, ya que es un mecanismo genérico y estándar, puede ser utilizado en casi cualquier otro lenguaje de programación.

Los algoritmos de ingresos y conexiones a base de datos, están encapsulados en el servicio web, de este modo, la conexión no puede ser sustraída y utilizada para fines adversos.

El uso de servicios web hace que nodos externos puedan comunicarse para obtener la información de los accesos, aumentando la escalabilidad de los sistemas al separar estos en nodos por si debe mejorar el rendimiento de las aplicaciones.

- No riesgos

Las conexiones que consultan los permisos no pueden ser sustraídas para fines adversos.

Tabla X. **Uso de patrón vista modelo controlador**

<b>Atributo</b>	Mantenibilidad.			
<b>Entorno</b>	Se requiere probar el funcionamiento de un algoritmo, se realiza una prueba unitaria, ya sea en el control o en la consulta.			
<b>Estímulo</b>	Un algoritmo no funciona como debe, y hay que verificarlo.			
<b>Respuesta</b>	Se realiza una instancia del objeto en un archivo de prueba y se envían los parámetros necesarios para su funcionamiento.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Programación convencional	S1		R1	

Continuación de la tabla X.

Uso de patrón MVC		T1 T2		R1
<b>Razonamiento</b>	S1. Disminuye mantenibilidad T1. Aumenta seguridad T2. Aumenta mantenibilidad R1. Código difícil de entender			

Fuente: elaboración propia con MS Word 2007.

Este escenario describe, cuando los algoritmos deben ser probados y analizados, enviando entradas sin necesidad de usar una vista especializada para ello, a esto se le llama, pruebas unitarias.

Las capas de las aplicaciones usan clases para la programación orientada a objetos, estas clases pueden ser instanciadas en archivos de prueba sin necesidad de copiar y pegar el algoritmo, proveyendo así, flexibilidad al realizar mantenimientos a las aplicaciones realizadas sobre la arquitectura.

Tanto en los archivos de control, como los archivos de consultas, pueden realizarse pruebas para verificar el funcionamiento de estos.

- Puntos de compensación (tradeoff)

Cuando se realizan pruebas, en programación estructurada es necesario copiar y pegar el algoritmo, posiblemente este pueda incluir alguna cadena de conexión. Si algún archivo de estos se olvida eliminarse, puede ser leído y su contenido sustraído para acciones malintencionadas.

El copiar y pegar algoritmos, llega a ser una tarea tediosa, además del riesgo de no poder sincronizar correctamente el código de prueba con el de producción.

- No riesgos

Al tener el código en archivos especializados, el riesgo de tener un algoritmo difícil de entender, se reduce, permitiendo a los desarrolladores, trabajar con mayor seguridad en su codificación.

Tabla XI. **Uso del patrón modelo vista controlador**

<b>Atributo</b>	Mantenibilidad.			
<b>Entorno</b>	Se requiere un algoritmo complejo, debe separarse en varias funciones en las distintas capas si lo requiere para funcionar.			
<b>Estímulo</b>	Un algoritmo utiliza muchos datos y operaciones.			
<b>Respuesta</b>	Se separa en operaciones básicas y se integra en un método.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Separación de tareas en capas		T1	R2	
Algoritmo en un solo archivo	S2		R1	
<b>Razonamiento</b>	S2. Disminuye mantenibilidad T1. Aumenta mantenibilidad R1. Codificación difícil de entender R2. Aplicación difícil de depurar			

Fuente: elaboración propia con MS Word 2007.

Este escenario muestra la situación, de un algoritmo complejo que debe realizarse, pero para ello se requiere separar su funcionalidad en tareas pequeñas e integrarse para obtener el resultado requerido.



Algunas tareas pueden llegar a ser muy complejas, en programación se utiliza el “divide y vencerás” para solucionarlas.

- Puntos de compensación (Tradeoff)

Al realizar una tarea grande en varias pequeñas, es más fácil de entender y realizar, los desarrolladores pueden incluso reutilizar partes de la tarea grande en otras tareas que requieran datos parecidos.

- Riesgos

Por estar separados los algoritmos, es un poco más difícil de depurar la aplicación, ya que cada tarea está dividida en capas, y encima estas capas podrían estar divididas en servidores distintos.

Tabla XII. **Uso de acceso a datos para consulta y transacciones**

<b>Atributo</b>	Disponibilidad.			
<b>Entorno</b>	Una consulta bloqueo alguna tabla y no deja realizar consultas a demás usuarios.			
<b>Estímulo</b>	Una consulta o transacción bloquea otras operaciones en curso.			
<b>Respuesta</b>	Se extrae el plan de ejecución de la consulta, se analiza y se reescribe para optimizarla.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso del plan de ejecución		T1	R1	
<b>Razonamiento</b>	T1. Aumenta mantenibilidad, disponibilidad y rendimiento R1. El rendimiento de la base de datos es afectado			

Fuente: elaboración propia con MS Word 2007.

Este escenario ejemplifica cuando una consulta es demasiado grande, y dificulta o bloquea el rendimiento y/o ejecución de otras. Una consulta que tarda

mucho, puede bloquear el mantenimiento de las tablas que se están consultando, disminuyendo así la disponibilidad de la aplicación.

- Puntos de compensación (Tradeoff)

La lectura y entendimiento del plan de ejecución provee al desarrollador de información para optimizar una consulta. El plan de ejecución da una idea de que puede estar funcionando mal y entonces corregirlo.

- Riesgo

El rendimiento de la base de datos puede ser afectado gravemente por una consulta mal diseñada, su optimización incrementará el rendimiento y un mejor manejo de recursos de la base de datos.

Tabla XIII. **Comunicación de la vista con el controlador por servicio web**

<b>Atributo</b>	Escalabilidad.			
<b>Entorno</b>	La carga de trabajo de un servidor es demasiado alta y afecta el rendimiento.			
<b>Estímulo</b>	Un servidor requiere demasiados recursos y requisiciones y se vuelve ineficiente.			
<b>Respuesta</b>	Se analiza que capa sobrecarga el servidor y se separa para dividir tareas.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso de interfaces para comunicar capas		T1	R1 R3	
Uso de instancias directas para comunicar capas	S1 S2		R2	

Continuación de la tabla XIII.

<b>Razonamiento</b>	S1. Disminuye portabilidad S2. Sincronización de librerías es más fácil T1. Aumenta portabilidad, escalabilidad R1. Sincronización de librerías no es bien realizada R2. La escalabilidad del sistema es más difícil R3. Operaciones del usuario interrumpidas
---------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fuente: elaboración propia con MS Word 2007.

Este escenario ejemplifica un servidor que está siendo sobrecargado por muchas llamadas al sistema o un incremento de usuarios, puede que este servidor provea tanto, servicios a usuarios como a otras aplicaciones y una sobrecarga afecta el rendimiento de este.

Puede que los recursos máximos de un servidor se vuelvan insuficientes para la carga de trabajo de este y debe utilizarse otro nodo, para separar tareas y mejorar los tiempos de respuesta de los servicios que provee.

La arquitectura provee la capacidad de separar capas para dividir las tareas de las aplicaciones en tantos nodos como se necesite, la comunicación de estos nodos son mediante servicios web, este método podría ser lento, pero en un análisis costo/beneficio, esta opción provee de portabilidad y estandarización para la comunicación entre nodos.

- Puntos de compensación (Tradeoff)

Al utilizarse servicios web, los nodos se vuelven portables, no es necesario utilizar el mismo lenguaje de programación para su comunicación, ni el mismo sistema operativo para su correcto funcionamiento.

Al poder separar en nodos, se incrementa la escalabilidad de los sistemas, no se requiere que un solo sistema contenga todas las capas de una aplicación, si una aplicación requiere más recursos que otras, se puede mover a un nodo distinto para mejorar el rendimiento y tiempos de respuesta.

- **Riesgos**

Los sistemas están compuestos por componentes, puede llegar el caso, que la actualización de algún componente, por error humano, no llegue a todos los nodos y se produzcan errores difíciles de entender y corregir.

Un riesgo es que al estar separados los nodos, puede que una capa no pueda comunicarse con otra, por un error en la red de internet o suministro eléctrico y que los sistemas no estén disponibles por un buen rato.

Tabla XIV. **Centralización de estilos CSS para las vistas de la aplicación**

<b>Atributo</b>	Usabilidad.			
<b>Entorno</b>	La GUI debe ser común en las aplicaciones para una adaptación rápida.			
<b>Estímulo</b>	Se requiere un cambio en la GUI para una vista más cómoda.			
<b>Respuesta</b>	Se modifica el estilo que las paginas implementan y actualiza todos los sistemas que la usan.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso de gestor de GUI		T1		
No utiliza gestor	S1		R1	
<b>Razonamiento</b>	S1. Disminuye mantenibilidad T1. Aumenta usabilidad R1. Estandarización de vistas se dificulta			

Fuente: elaboración propia con MS Word 2007.

Este escenario describe lo importante que es mantener un estándar en las vistas del usuario y lo flexible que debe ser a cambios.

En el Registro General de la Propiedad se tiene mucha gente mayor trabajando para la institución, estas personas son muy resistivas al cambio, e incluso muchas optan por no usar las herramientas que provee el departamento de informática, alguna de las razones es que les confunde utilizar distintos sistemas con distintas vistas en la presentación de formularios y datos.

Mantener una GUI consistente es importante manteniendo un estándar en las vistas de las aplicaciones, de este modo, los usuarios no tendrán que adaptarse a un sistema distinto de nuevo.

Algunas de las quejas es la combinación de colores en los sistemas, estos deben cambiarse eficaz y rápidamente según los requerimientos de los usuarios, y utilizar hojas de estilo estáticas podría dificultar esta habilidad.

- Puntos de compensación (Tradeoff)

El uso de un gestor de vistas aumenta la mantenibilidad de las aplicaciones, un par de clicks es necesario para cambiar el estilo o el color de los sistemas y no se requiere modificar el código fuente.

Al tener una GUI consistente la adaptación de un sistema a otro es más fácil que estar aprendiendo a utilizar de nuevo vistas distintas.

### 8.3. Testeo

Una vez realizada la evaluación de los escenarios, se crea una lluvia de ideas, donde se crean nuevos ambientes que pudieron ser obviados en los pasos anteriores, se evalúan y se documentan en las últimas dos fases de la evaluación ATAM.

#### 8.3.1. Lluvia de ideas y priorización de escenarios

A continuación se presentan escenarios que fueron obviados en los pasos anteriores para ser analizados y priorizados.

Tabla XV. **Tabla de escenarios no considerados**

<b>Atributo de calidad</b>	<b>Seguridad</b>
Propiedad del atributo	Sistemas comparten información al comunicarse por internet.
Escenario	1. Los sistemas deben utilizar conexión segura para compartir información en la red (H,H).
Escenario	2. El sistema de usuarios deben acoplar variados tipos de usuarios, no solo los internos de la institución para su autenticación en los sistemas (H, M).
<b>Atributo de calidad</b>	<b>Mantenibilidad</b>
Propiedad del atributo	Los sistemas cambian en el transcurso del tiempo.
Escenario	3. Una característica que ya no se necesitaba, se elimino pero ahora se necesita de nuevo (M,M).
Escenario	4. Una versión nueva de una aplicación no funciona correctamente (H,H).

Continuación de la tabla XV.

Atributo de calidad	Disponibilidad
Escenario	5. Los datos del sistema no están disponibles (H,H).
Escenario	6. No se pueden subir archivos por medio del sistema (M,M).

Fuente: elaboración propia con MS Word 2007.

Basadas en la experiencia en la institución y las opiniones basadas de los usuarios, se obtuvieron los siguientes escenarios como principales a analizar.

Tabla XVI. **Tabla de escenarios de mayor relevancia**

1	Una institución necesita consultar datos del Registro General de la Propiedad.	Producción	Se utiliza HTTPS para la conexión segura para la transmisión de información.
4	El usuario no obtiene los mismos resultados que la versión de una aplicación anterior.	Producción	Se hace una desactualización de la aplicación que da problemas.
5	El servidor de base de datos se cae por fallas en la red o energía eléctrica.	Producción	Se configura un servidor que replique la información del servidor original para no perder disponibilidad de datos.

Fuente: elaboración propia con MS Word 2007.

### 8.3.2. Análisis de propuestas arquitectónicas

Análisis de las ventajas de los atributos no tomados en cuenta anteriormente, identificando riesgos que sucedan o puedan suceder.

Tabla XVII. **Servicios web de comunicación entre sistemas internos y externos**

<b>Atributo</b>	Seguridad.			
<b>Entorno</b>	Los sistemas deben utilizar conexión segura para compartir información en la red.			
<b>Estímulo</b>	Una institución necesita consultar datos del Registro General de la Propiedad.			
<b>Respuesta</b>	Se utiliza HTTPS para la conexión segura para la transmisión de información.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso de servicio web con HTTPS	S1	T1 T2		R1
Uso de servicio web sin HTTPS	S2		R1	
<b>Razonamiento</b>	S1. Disminuye rendimiento S2. Disminuye seguridad T1. Aumenta seguridad T2. Aumenta interoperabilidad R1. Los datos pueden ser interceptados y manipulados			

Fuente: elaboración propia con MS Word 2007.

Este escenario ejemplifica la situación en donde los sistemas se comunican a través del internet.

La internet es un conjunto de redes interrelacionadas donde la información viaja hacia su destino y mucha información viaja sin encriptación, pudiendo así, ser capturada, manipulada y reenviada, credenciales, información personal, datos confidenciales son capturados para ser analizados y sacar provecho de estos.

Las conexiones seguras evitan estos ataques, mediante protocolos de seguridad que utiliza criptografía SSL y TLS, estándares utilizados en internet, para el envío de información confidencial.



- Puntos de sensibilidad (Sensitivity)

La codificación y descodificación produce trabajo extra en los servidores web, tanto en páginas de usuarios como en servicios web, reduce el rendimiento al apartar recursos para este proceso.

- Puntos de compensación (Tradeoff)

Los datos confidenciales no deben estar disponibles a ningún tipo de personas ajenas a la institución, el uso de criptografía evita la usurpación de identidad de las personas, evitando así, pérdidas monetarias o información clasificada que no debe estar disponible al público tan fácilmente.

Al utilizar conexiones seguras, aumenta la confiabilidad del uso de la información, por lo tanto la interoperabilidad aumenta, ya que esta información puede ser compartida a empresas o instituciones del gobierno sin temor al robo de información.

- No riesgos

Los datos al estar encriptados, no será posible o será muy difícil robar información confidencial.

Tabla XVIII. **Versiones de aplicación**

<b>Atributo</b>	Mantenibilidad.
<b>Entorno</b>	Una versión nueva de una aplicación no funciona correctamente.
<b>Estímulo</b>	El usuario no obtiene los mismos resultados que la versión de una aplicación anterior.

Continuación de la tabla XVIII.

<b>Respuesta</b>	Se hace una desactualización de la aplicación que da problemas.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso de versionamiento de aplicaciones	S1			R1
<b>Razonamiento</b>	S1. Aumenta mantenibilidad. R1. La información mostrada o enviada puede no ser la correcta en cierto tiempo.			

Fuente: elaboración propia con MS Word 2007.

Este escenario describe cuando, los sistemas son actualizados por versiones mejoradas de los mismos.

Los sistemas mejoran continuamente, ya sea en actualizaciones que mejoren el rendimiento o la capacidad de estos, corrijan defectos o insuficiencias de estos, o bien, por la creación de requerimientos nuevos o requerimientos que ya no se necesitan y deban ser eliminados.

Puede llegar el caso que una aplicación nueva, no funcione como debería, mostrando información errónea que la versión anterior si mostraba correctamente, o bien, algún algoritmo que se dejó de utilizar, ahora se necesita de nuevo. El uso del versionamiento provee de seguridad en el código fuente, dando un historial de las versiones estables de las aplicaciones, para así, si una aplicación no funciona como debería, se baja de versión mientras se corrige la nueva, o bien, se recupera código perdido que anteriormente se utilizaba.

- Puntos de sensibilidad (Sensitivity)

Al tener un historial, se mejora el mantenimiento del código, todos los cambios efectuados, tanto mejoras, eliminación o agregación de código, se encuentra en un repositorio donde se puede navegar y clasificar la codificación de las aplicaciones.

- No riesgos

Si una aplicación necesita ser cambiada a una versión anterior, se explora el repositorio de código y se extrae para ser desplegada y no perder tiempo en el proceso de actualización mientras se arreglan los errores detectados.

Tabla XIX. **Replicación de datos**

<b>Atributo</b>	Disponibilidad.			
<b>Entorno</b>	Los datos del sistema no están disponibles.			
<b>Estímulo</b>	El servidor de base de datos se cae por fallas en la red o energía eléctrica.			
<b>Respuesta</b>	Se configura un servidor que replique la información del servidor original para no perder disponibilidad de datos.			
<b>Decisión arquitectónica</b>	Sensitivity	Tradeoff	Riesgo	No Riesgo
Uso de mecanismo de replicación	S1	T1	R1	R2
<b>Razonamiento</b>	S1. Disminuye rendimiento. T1. Aumenta disponibilidad. R1. Una mala configuración puede que la replicación no sea correcta. R2. Los usuarios no tienen los datos para hacer mantenimientos o consultas.			

Fuente: elaboración propia con MS Word 2007.

Este escenario ilustra si el modelo o bien, los datos, no están disponibles para ser consultados o darles mantenimiento.

Las aplicaciones son una interfaz de comunicación de los datos hacia el usuario, estas crean representaciones gráficas de los datos para ser manipulados y visualizados, si los datos no están disponibles, los sistemas tampoco, ya que no hay datos de referencia que manipular.

Los mecanismos de alta disponibilidad entran en este juego, entre estos existe la replicación de datos, un servidor principal provee servicios de datos, mientras otro secundario replica las operaciones que el servidor principal está manejando. Si hubiera una falla eléctrica, del proveedor de internet o un posible desastre natural sobre el servidor principal, se tiene el secundario que esta replicando la información se convierte en principal y como los datos están sincronizados, la pérdida debe ser mínima.

- Puntos de sensibilidad (Sensitivity)

El mecanismo de replicación consume recursos de los servidores, puede afectar el rendimiento de los sistemas.

- Puntos de compensación (Tradeoff)

Si los sistemas están disponibles, los usuarios siempre podrán trabajar sobre los datos, evitando así posibles caídas de información que puede conllevar a pérdidas monetarias a la institución.

- Riesgos

Una mala configuración puede provocar una mala sincronización entre servidores, la persona o personas encargadas de esta tarea debe estar segura de que los mecanismos de replicación sean efectivos ante los posibles incidentes que puedan ocurrir.

- No riesgos

Al tener alta disponibilidad la pérdida de datos se minimiza y la disponibilidad de los sistemas aumenta al prevenir dichos eventos

#### **8.4. Reporte**

Se presenta un resumen de la evaluación ATAM, presentando la cantidad de puntos de sensibilidad, concesión, riesgos y prevenciones de riesgos, provee información de posibles problemas arquitectónicos y sus posibles soluciones.

##### **8.4.1. Resultados de análisis**

Como resultado de la aplicación del método, se obtuvo una lista de 14 escenarios priorizados y las diferentes propuestas arquitectónicas que los intentan satisfacer, a partir de las cuales se descubrieron 6 puntos de sensibilidad, 18 de concesión o compensación (tradeoff), 8 riesgos asociados y 9 prevenciones de riesgo. Las propuestas arquitectónicas son una excelente guía para tomar futuras decisiones arquitectónicas, ya que se analizó su impacto en la arquitectura en cuestión y atacan las expectativas de calidad que la institución tiene de los sistemas que lo implementan.

Por ser una arquitectura nueva que evolucionó en el transcurso del tiempo que fue desarrollado, el sistema de usuarios y permisos, no contaba con documentación o especificaciones bien definidas, más bien, sus componentes originales estaban altamente acopladas, pero cuando se tuvo que desarrollar un segundo sistema, se vió la oportunidad de aprovechar estos componentes para este siguiente proyecto, en ese momento se empezó a desarrollar como arquitectura como tal, teniendo al sistema de usuarios y permisos parte de esta misma.

Los puntos de sensibilidad que se extrajeron del análisis, afectan principalmente el rendimiento de las aplicaciones, aunque el impacto de este es no es significativo y de acuerdo con los objetivos de la institución, es permisible dicho impacto, además, la institución invierte mucho en tecnología, manteniendo así, la infraestructura de hardware actual y optimizada, este bajo impacto es compensado con el hardware de la institución. Uno de las posibles mejoras a manejar respecto este punto, es la utilización de aplicaciones que monitoreen automáticamente el uso de recursos y análisis de estos.

Los puntos de compensación proveen ventajas que deben aprovecharse en la utilización de la arquitectura, estas mejorarán el proceso de desarrollo, mantenimiento, seguridad, disponibilidad del software, muchas tareas y deficiencias que conlleva la programación tradicional son mejoradas y simplificadas por las herramientas y APIs de la arquitectura.

Un riesgo importante a mencionar, es la posible falta de disponibilidad del sistema de usuarios y permisos, ya que este es crucial para el funcionamiento de la arquitectura, pues controla la autenticación y accesos a los demás sistemas. Si este en algún momento dejara de funcionar, todos los demás sistemas dejarían de estar disponibles. Para mitigar esto, el sistema cuenta con

los componentes propios de la arquitectura que protegen de ataques como inyección SQL o XSS. Otra propuesta para mitigar este riesgo, es el desarrollo de un componente que provea de funcionalidades mínimas en los sistemas, si en dado caso el sistema de usuarios y permisos no esté disponible.

Los recursos de hardware no son infinitos y es un riesgo que estos puedan agotarse. Se debe contar con la capacidad del monitoreo automático de recursos, ya sea propio de la arquitectura o por aplicaciones de terceros.

Entre los riesgos identificados, existen los provocados por error humano, como la mala configuración del uso de la arquitectura y/o del hardware, pero dado el conocimiento de estos, se pueden prevenir y mitigar al usar esta documentación como referencia. Los puntos de apoyo (no riesgos) dan el conocimiento de qué se estará previniendo al utilizar las propuestas arquitectónicas de la arquitectura SKRN y así aprovechar el máximo potencial de esta.

Existen riesgos que afectan a la arquitectura que no están como responsabilidad del equipo de desarrollo, pero que es importante tenerlos en cuenta, son los riesgos de infraestructura, como la conexión a energía eléctrica o la disponibilidad de los enlaces de red. Otros riesgos son los ambientales, como desastres naturales, que no se pueden predecir, pero que se pueden prevenir, al tener los servidores en un lugar seguro, que prevenga incendios o destrucción de los servidores físicos o bien, la replicación de datos y codificación.

La arquitectura SKRN provee de muchas y poderosas herramientas, mitiga riesgos, provee muchos atributos de calidad como lo es la seguridad, mantenibilidad, escalabilidad, modificabilidad, un API completo para el manejo

de usuarios y permisos mediante el sistema con el mismo nombre y herramientas que facilitan la generación de documentación. Su utilización facilita la codificación y su bajo acoplamiento permite la implementación o sustitución de los módulos que lo conforman.

A partir de este análisis se concluyen que, algunas características arquitectónicas deben mejorarse:

- Seguridad de comunicación por internet: la utilización de HTTPS es crucial para esta característica, evitando de este modo, usurpación de datos y/o identidad.
- Monitoreo automático de recursos: se recomienda utilizar aplicaciones de terceros que revisen y notifiquen posibles problemas de recursos de los sistemas, los proveedores del software utilizado en la institución como de DBMS MySQL o de las máquinas virtuales VMWare, proveen de estos sistemas.
- Correcta actualización de componentes: al actualizar un componente, este debe ser aplicado en cada nodo en que se esté implementando, una mala actualización tiene el riesgo que los sistemas no funcionen correctamente.
- Diseño y validación de consultas: una consulta mal hecha, crea un plan de ejecución de bajo rendimiento, impactando considerablemente los sistemas y consumiendo recursos innecesarios.



- Funcionamiento mínimo de sistemas: si en algún momento el sistema de seguridad no está disponible, se recomienda desarrollar un componente que provea de una funcionalidad mínima en los sistemas que dependan de este.

Conociendo y mitigando estos riesgos, se protege el sistema informático, por lo tanto, también los objetivos del negocio de la institución, concluyéndose que, la arquitectura SKRN es apta para su utilización en el Registro General de la Propiedad y cualquier otra institución o empresa que desee implementarla.



## CONCLUSIONES

1. Una arquitectura de software estandariza la codificación, permitiendo un desarrollo y mantenimiento rápido de aplicaciones, ya que se rige por reglas de cómo el flujo de información debe viajar por la aplicación, como se debe estructurar y especializar en cada archivo. Esta es una ventaja para el desarrollador, pues si necesita corregir un problema, encontrarlo lleva menos tiempo, pues todos los sistemas siguen el mismo patrón, incluso si el sistema no fue desarrollado por él.
2. Una institución gubernamental como lo es el Registro General de la Propiedad, le es factible utilizar una la arquitectura para la modernización tecnológica, el tiempo de desarrollo, costos y la complejidad de la aplicación disminuye y aumenta la calidad del software. Una aplicación basada en la arquitectura SKRN, economiza el tiempo del desarrollo del gestor de usuarios, acceso y manipulación de datos, configuración de protección de información, generación de roles y permisos a vistas, diseño y modificación de estas, presentación y generación de manuales de usuarios y técnico, enfocando al desarrollador exclusivamente al propósito del proyecto, reduciendo, en promedio, un 66 % del esfuerzo y tiempo si usara programación sin arquitectura definida.

3. El uso de propuestas arquitectónicas tales como la implementación de patrones de diseño, centralización de usuarios y permisos, comunicación por medio de servicios, previenen deficiencias en la codificación convencional y ofrecen seguridad adicional e incrementan la facilidad de alcanzar los atributos de calidad requeridos, además, facilita la verificación de calidad del código de la aplicación.
4. La evaluación de la arquitectura permite identificar qué deficiencias, ventajas y riesgos tiene esta, cómo mitigarlas y prepararse ante posibles eventualidades que surjan en el transcurso del tiempo. También, permite que las propuestas arquitectónicas implementadas y configuradas sean evaluadas y tomar las mejores decisiones que se necesiten en la institución de acuerdo a sus requerimientos.
5. La evaluación ATAM, concluyó que la arquitectura SKRN es buena para su seguimiento e implementación en los sistemas del Registro General de la Propiedad, ya que reduce los costos y tiempos del desarrollo de software. Como se vió en el capítulo 8, el impacto para alcanzar los atributos de calidad requeridos es alto y la cantidad de riesgos al utilizarla es mínima, limitando estos a infraestructura y configuración.
6. Se mejoró el conocimiento de la arquitectura en cuestión, ya que al ser evaluada, se encontraron deficiencias a corregir y se conocieron mejor sus fortalezas. Un sistema informático siempre estará evolucionando en el tiempo, la arquitectura misma también lo puede hacer, mejorando así sus capacidades para fortalecer y/u obtener atributos de calidad en la institución, de igual modo reducirá la cantidad de posibles riesgos que se hayan o puedan identificar más adelante.

## RECOMENDACIONES

1. Se debe utilizar ingeniería de software en las instituciones gubernamentales para mejorar el proceso de desarrollo de aplicaciones. No se debe menospreciar la capacidad que tiene la tecnología de la información en una institución y su inversión es tan importante como la de procesos contables, jurídicos, de infraestructura, entre otros. El recurso humano usado en el desarrollo también lo es, pues estos dan seguimiento inmediato a las aplicaciones desarrolladas, es más costoso que una persona nueva aprenda la arquitectura, que mejorar la situación laboral de alguien con experiencia sobre esta.
2. El uso de Interfaces de programación de aplicaciones o APIs deben priorizarse, pues facilita el desarrollo de software, además incrementa la calidad, reutilización y estandarización del código fuente, además, muchos componentes, según sus características, mejoran los atributos de calidad de las aplicaciones. Es menos costoso obtener un componente probado y documentado, que crear uno nuevo, esto solo se hace si los componentes disponibles en el mercado no llenan los requerimientos necesarios para la aplicación.
3. Se debe usar una arquitectura de software robusta que cubra la mayoría de las necesidades de una institución. Además, facilita la obtención de atributos de calidad ya que esta las integra en sí misma. Una mala elección conlleva a perder tiempo y dinero a la institución, cuando ésta ya se ha implementado en aplicativos de software, es mejor invertir tiempo en

la evaluación y testeo de una arquitectura o un componente, que hacer una actualización masiva del software.

4. Se deben fomentar las buenas prácticas del desarrollo de software, tales como el uso de patrones de diseño, normalización en la base de datos, bajo acoplamiento y alta cohesión de componentes y demás, pues ayuda a la transparencia de los procesos y a la disminución de la corrupción por medio de la informática. La ingeniería del software provee de esta característica, ya que contiene principios y metodologías para aplicaciones rentables y fiables. Las malas prácticas como por ejemplo, el no utilizar índices o no usar integridad referencial, hacen software de mala calidad, lento y propenso a errores, disminuyendo así el alcance de los atributos de calidad requeridos y por consiguiente no alcanzar los objetivos del negocio.
5. Es necesario utilizar una arquitectura de software para el desarrollo, pero esta debe ser evaluada y documentada para futuros cambios en los sistemas, la mala elección de esta, puede llevar a costes innecesarios, tanto administrativos como de tiempo y dinero, una arquitectura debe proveer de soluciones para problemas que existieron, existen y que posiblemente puedan existir en un futuro.
6. La arquitectura en cuestión y/o de infraestructura, se debe evaluar periódicamente, al menos, una vez al año, para identificar posibles riesgos y problemas de implementación, se debe verificar el cumplimiento de los atributos de calidad de la institución, ya que estos, al basarse en los objetivos del negocio, no es permisible perderlos.

## BIBLIOGRAFÍA

1. BOS, Bert. *Cascading Style Sheet* [en línea]. [ref. de 21 de junio de 2013]. Disponible en Web <<http://www.w3.org/Style/CSS/>>.
2. CARDOSO M., Lucía I. *Sistemas de base de datos II, teoría aplicada para profesores y estudiantes*. Caracas, Venezuela: Universidad Católica Andrés Bello, 2006. 205 p.
3. CORTEZ MORALES, Roberto. *Introducción al análisis de sistemas y la ingeniería de software*. San José, Costa Rica: Universidad Estatal a Distancia, 1998. 168 p.
4. FREEMAN, Elisabeth; FREEMAN, Eric; BATES, Bert. *Head first Design patterns*. USA: O'Reilly Media, 2004. 678 p.
5. *HTML* [en línea]. [ref. 18 de noviembre de 2011]. Disponible en Web: <<http://www.w3.org/community/webed/wiki/HTML>>.
6. *JavaScript* [en línea]. [ref. de 17 de noviembre de 2012]. Disponible en Web: <<https://developer.mozilla.org/es/docs/JavaScript>>.
7. MARQUÉZ SOLIS, Santiago. *La web semántica*. Madrid: Universidad Politécnica de Madrid, 2007. 195 p.
8. PONS CAPOTE, Olga. *Introducción a los sistemas de bases de datos*. Madrid: Paraninfo, 2008. 312 p.

9. *¿Qué es CSS?* [en línea] <<http://es.html.net/tutorials/css/lesson1.php>> [Consulta: 21 de junio de 2013].
10. *¿Qué es un patrón de diseño?* [en línea]. <<http://msdn.microsoft.com/es-es/library/bb972240.aspx>> [Consulta: 23 de junio de 2013].
11. RAMOS SALAVERT, Isidro; LOZANO PÉREZ, María Dolores. *Ingeniería del software y bases de datos: tendencias actuales*. Ciudad Real, España: Universidad de Castilla-la Mancha, 2000. 262 p.
12. SOLANA, Aroa. *Windows Communication Fundations 4.0*. Madrid: Luarna, 2011. 1380 p.
13. SOMMERVILLE, Ian. *Ingeniería del Software*. 7a ed. Madrid: Pearson Education, 2005. 712 p.
14. WILISON, Simon. *A Re-introduction to JavaScript* [en línea]. [ref. 7 de marzo de 2006]. Disponible en Web: <[https://developer.mozilla.org/en-US/docs/Web/JavaScript/A\\_re-introduction\\_to\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript)>.