

# Universidad de las Ciencias Informáticas

## Facultad 3



### **Título: Propuesta de Arquitectura Orientada a Servicios para el Módulo de Inventario del ERP Cubano**


TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERÍA EN CIENCIAS  
INFORMÁTICAS

**Autores:** Enrique Chaviano Gómez  
Yoan Arlet Carrascoso Puebla

**Tutores:** Ing. José Raúl Perera Morales  
Ing. Diosmani Meriño Hechavarría

**Tutor asesor:** MSc. José Raúl Rodríguez Galera

Ciudad de la Habana  
Junio 2008

An open book with aged, yellowed pages is shown from a slightly elevated angle. The book is open to two pages, and the text is overlaid on the right page. The background is a soft, out-of-focus white.

*“Investigar es ver lo que todo el mundo ha visto, y pensar lo que nadie más ha pensado.”*

*Albert Szent Gyorgi*

## **DECLARACIÓN DE AUTORÍA**

Por este medio declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los \_\_\_\_ días del mes de junio del año 2008.

### **Autor (es)**

Enrique Chaviano Gómez

---

Yoan Arlet Carrascoso Puebla

---

### **Tutor (es)**

Ing. José Raúl Perera Morales

---

Ing. Diosmani Meriño Hechavarría

---

## **DATOS DE CONTACTO**

**Tutor Ing. José Raúl Perera Morales.** Graduado de Ingeniero en Ciencias Informáticas en la UCI en el 2007. A partir de ese momento se incorpora como Asesor del Proyecto Convenio Cuba Venezuela y actualmente trabaja como Jefe de Dpto. de Técnicas de Programación. En su trabajo de diploma investigó sobre las arquitecturas de software haciendo una propuesta para un sistema de inventario.

**Tutor Ing. Diosmani Meriño Hechavarría.** Graduado de Ingeniería Informática en la Universidad de Holguín “Oscar Lucero Moya” en el 2005. Desde el año 2005 ha trabajado en Sistemas de Gestión de Inventarios para Soluciones de Gestión Empresarial. En su trabajo de diploma presentó TRANSOLVER, un sistema de para problemas de distribución y asignación de recursos. Ha sido tutor de 4 tesis de diploma. Actualmente se desempeña como subgerente de desarrollo en DESOFT en la provincia de Villa Clara.

**Tutor asesor MSc. José Raúl Rodríguez Galera.** Licenciado en Educación. Pedagogo. Máster en Ciencias en Género, Educación Sexual y Salud Reproductiva. Profesor de Metodología de la Investigación Científica en pregrado y postgrado. Cuenta con 24 años de experiencia en la docencia. 8 años de experiencia en tutorías, oponencias y tribunales de tesis de grado. Actualmente cotutorea dos tesis de maestría en Farmacología en la UH y 16 de grado para Ingenieros en Ciencias Informáticas en la UCI. Cuenta en su haber con investigaciones institucionales y nacionales, participación en eventos nacionales, internacionales y un mundial, 10 publicaciones en los últimos 5 años. Ex-vice decano de Investigaciones del ISCM de La Habana.

## **AGRADECIMIENTOS**

Primero que todo queremos agradecer al compañero Fidel Castro Ruz y a la Revolución por permitirnos ser parte de este proyecto y esta justa sociedad...

A nuestros padres por la confianza, el apoyo y el amor, por sus consejos, por ser nuestros principales guías y ayudarnos a llegar hasta aquí. ¡Nunca los defraudaremos!

A Diosmani y José Raúl nuestros tutores y amigos, por las ideas, el apoyo, confianza, y paciencia.

A Jose Raúl Galera por brindarnos su apoyo, su tiempo y experiencia Metodológica.

A René por apoyo y visión crítica, por brindarnos sus conocimientos.

A Frank por escucharnos y brindarnos sus ideas.

A nuestras novias por estar siempre a nuestro lado, dedicándonos confianza y apoyo, día a día, ¡noche a noche!

A los amigos que siempre nos escucharon, nos dieron ideas bien críticas que nos sirvieron de mucho y su apoyo.

A todos aquellos que de una forma u otra colaboraron con nosotros para la confección de este trabajo de diploma.

## **DEDICATORIA**

A mi familia por todo el cariño y apoyo que siempre me han dado, mis padres, mi hermana, en especial a mis abuelos.

*Enrique Chaviano Gómez*

A mi familia por su confianza, apoyo y cariño, y en especial a mis padres que son mi orgullo y el alma que me impulsa a seguir adelante.

*Yoan Arlet Carrascoso Puebla*

## **RESUMEN**

En el presente trabajo de diploma se expone una propuesta de Arquitectura Orientada a Servicios para el Módulo de Inventario del ERP Cubano, de modo que le posibilite al mismo ser: portable, seguro, escalable e integrable con otros sistemas y que a la vez sea extensible a otros módulos del ERP. Para lograr el objetivo propuesto de definir y describir la arquitectura de software se partió de un estudio de los elementos que constituyen la misma y las tendencias actuales según los autores más prestigiosos del tema, así como la experiencia del desarrollo del Sistema de Gestión de Inventarios y Almacenes (SIGIA) desarrollado en la facultad 3.

El éxito de la definición y descripción de la arquitectura estará en dependencia principalmente de las decisiones arquitectónicas que se tomen durante su desarrollo, la tecnología más adecuada que se decida y aspectos del diseño que permitan un desarrollo favorable del propio sistema, donde se tengan en cuenta las relaciones y configuraciones entre los componentes y que permita organizar la estructura del sistema que se desarrolla brindando a todo el equipo de trabajo una idea clara de lo que se esta desarrollando.

## **PALABRAS CLAVES**

Arquitectura de Software, estilo, patrón, componente, framework, *Service Oriented Architecture* (SOA), servicio web, integración, *Enterprise Resource Planning* (ERP), XML, SOAP, UDDI, WSDL.

## TABLA DE CONTENIDO

DATOS DE CONTACTO .....	2
AGRADECIMIENTOS.....	3
DEDICATORIA .....	4
RESUMEN.....	5
INTRODUCCIÓN.....	1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.....	5
1.1. INTRODUCCIÓN.....	5
1.2. ESTADO DEL ARTE DE LA ARQUITECTURA DE SOFTWARE .....	5
1.2.1. <i>Principales definiciones</i> .....	5
1.2.1.1. Arquitectura de Software.....	5
1.2.1.2. Estilo.....	7
1.2.1.3. Patrón.....	8
1.2.1.4. Marco de trabajo .....	9
1.2.2. <i>Tendencias</i> .....	10
1.2.3. <i>Diferencia entre Arquitectura de Software y Diseño</i> .....	12
1.3. NIVELES DE ABSTRACCIÓN DE LA ARQUITECTURA DE SOFTWARE .....	13
1.3.1. <i>Estilos Arquitectónicos</i> .....	13
1.3.1.1. Clasificación de Estilos.....	13
1.3.1.2. Estilos más Populares.....	15
1.3.2. <i>Patrones Arquitectónicos</i> .....	21
1.3.3. <i>Relación entre los Niveles de Abstracción</i> .....	22
1.3.4. <i>Marcos de Trabajo</i> .....	23
1.4. SOA: DESAFÍO DEL PRESENTE .....	24
1.4.1. <i>Razones para usar SOA</i> .....	24
1.4.2. <i>Elementos de SOA</i> .....	25
1.4.2.1. Tipos de Servicios .....	27
1.4.3. <i>Servicios Web</i> .....	28
1.4.3.1. Tecnologías asociadas.....	29
1.4.4. <i>Composición de Servicios</i> .....	31
1.4.5. <i>SOA &amp; Web Service</i> .....	32
1.4.6. <i>Modelo de Madurez de SOA</i> .....	33
1.5. ADLS Y UML .....	33
1.6. CALIDAD EN LA ARQUITECTURA DE SOFTWARE.....	35
1.7. CONCLUSIONES PARCIALES.....	38
CAPÍTULO 2: PROPUESTA DE SOLUCIÓN.....	39
2.1. INTRODUCCIÓN.....	39
2.2. DESCRIPCIÓN DE LA ARQUITECTURA.....	39
2.2.1. <i>Introducción</i> .....	39
2.2.1.1. Propósito .....	39
2.2.1.2. Alcance.....	39
2.2.2. <i>Representación Arquitectónica</i> .....	40
2.2.3. <i>Objetivos y Restricciones Arquitectónicas</i> .....	50
2.2.4. <i>Tamaño y Rendimiento</i> .....	55
2.2.5. <i>Vista de Casos de Uso</i> .....	58



2.2.6.	<i>Vista Lógica</i> .....	60
2.2.7.	<i>Vista de Despliegue</i> .....	66
2.2.8.	<i>Vista de Implementación</i> .....	69
2.2.9.	<i>Vista de Datos</i> .....	71
2.2.10.	<i>Flujo de Trabajo</i> .....	72
2.2.11.	<i>Conclusiones Parciales</i> .....	75
CAPÍTULO 3. AMBIENTE DE DESARROLLO Y CALIDAD EN LA ARQUITECTURA .....		76
3.1.	INTRODUCCIÓN .....	76
3.2.	AMBIENTE DE DESARROLLO .....	76
3.2.1.	<i>Herramientas Horizontales</i> .....	76
3.2.1.1.	Sistema Operativo .....	76
3.2.1.2.	Antivirus .....	77
3.2.1.3.	Gestión de Recursos .....	78
3.2.1.5.	Gestión de Proyecto .....	80
3.2.1.6.	Gestión Documental .....	81
3.2.1.7.	Seguimiento de Errores .....	82
3.2.2.	<i>Herramientas Verticales</i> .....	83
3.2.2.1.	Herramientas de Modelado .....	83
3.2.2.2.	Máquina Virtual de Java .....	84
3.2.2.3.	Entornos de desarrollo integrados (IDE) .....	85
3.2.2.4.	Lenguaje de Programación .....	86
3.2.2.5.	Frameworks .....	87
7.1.1.1.	Servidores de Aplicaciones .....	97
7.1.1.2.	Sistema Gestor de Base de Datos .....	99
7.2.	CALIDAD DE LA ARQUITECTURA .....	101
7.3.	VENTAJAS Y DESVENTAJAS DE LA ARQUITECTURA PROPUESTA .....	105
7.4.	VALORACIÓN DE LA SOLUCIÓN POR PARTE DE LOS AUTORES .....	108
7.5.	VALORACIÓN DE ESPECIALISTAS .....	109
7.5.1.	<i>Premios otorgados a la solución</i> .....	111
7.6.	CONCLUSIONES PARCIALES .....	111
CONCLUSIONES .....		112
RECOMENDACIONES .....		113
REFERENCIAS BIBLIOGRÁFICAS .....		115
BIBLIOGRAFÍA .....		119
ANEXOS .....		¡ERROR! MARCADOR NO DEFINIDO.
GLOSARIO .....		¡ERROR! MARCADOR NO DEFINIDO.

## INTRODUCCIÓN

Con la desaparición del socialismo en la Unión de Repúblicas Socialistas Soviéticas y el recrudecimiento del bloqueo de Estados Unidos hacia Cuba en la década de 1990, la economía cubana quedó paralizada. Debido a esto el país se vio en la necesidad de realizar un proceso de rediseño de la política económica. Desde entonces se comenzó a llevar a cabo un programa gradual de medidas económicas que tenían como objetivo superar la situación de la nación al menor costo social posible.

A raíz de las dificultades económicas a las que se estaba enfrentando el país, el estado cubano comenzó a invertir en varios sectores estratégicos de la economía y empezó a destinar gran cantidad de recursos a elevar el nivel de informatización de la sociedad, donde el ejemplo más fehaciente y actual lo constituye la creación de la Universidad de las Ciencias Informáticas para producir software de gran calidad e insertarlo en el mercado mundial. Como resultado de dicha estrategia, se han estado obteniendo sistemas empresariales más eficientes y de mayor calidad, a la altura de las exigencias tecnológica del momento.

A partir del perfeccionamiento empresarial, las empresas trabajan para lograr una mayor eficiencia en su producción o en los servicios que brindan. De ahí la utilización de sistemas informáticos que permitan la automatización de los procesos empresariales y de este modo, hacerlos mucho más eficientes. Uno de los sistemas que posibilita esto, son los *Enterprise Resources Planning* (ERP), que son sistemas de gestión de información que integran y automatizan en una única aplicación todos los procesos de negocio de una empresa, entiéndase por esto: producción, ventas, compras, logística, contabilidad, gestión de proyectos, inventarios, control de almacenes, pedidos, nóminas, recursos humanos, entre otras.

Los ERP brindan soporte a los usuarios del negocio, un manejo eficiente de la información para la toma de decisiones, tiempos rápidos de respuesta y una disminución de los costos totales de las operaciones. (Martínez, et al., 2001).

Uno de los procesos más importantes en la planificación de recursos de una empresa son los sistemas de inventario y por ello constituye una base sólida en el desarrollo de sistemas ERP.

En la economía de hoy día, el manejo de inventarios ha llegado a la cumbre de los problemas de la administración de empresas debido a que es un componente fundamental de la productividad. Si se mantienen inventarios demasiado altos, el costo podría llevar a una empresa a tener problemas de liquidez financiera. Por otro lado, si se mantiene un nivel insuficiente de inventario, podría no atenderse

a los clientes de forma satisfactoria, lo cual traería como consecuencia reducción de ganancias y pérdida de mercado (Sanchez, 2006). Por lo que se puede apreciar que llevar inventarios sanos garantizará a cualquier empresa e industria una mayor confiabilidad a la hora de mover los productos dentro del almacén.

En Cuba existen Sistemas de Gestión de Inventarios, entre ellos se tienen: *El Sistema de Inventario del Patrimonio Cultural y Natural (SIP)*, como su nombre lo indica fue creado con fines patrimoniales como: el registro, el control y la conservación de los mismos. *El Sistema Informatizado para el Procesamiento del Inventario Forestal (INVENFOR 1.0)*, realizado con el fin de registrar y procesar los datos obtenidos del inventario forestal en función de estimar los parámetros dasométricos de los Rodales de forma tal que tribute a una planificación del manejo forestal más eficiente. Otros ejemplos de sistemas son: *DRIM, FONDUS, SI e IHMM 2000* de la empresa *GET (Grupo de Electrónica para el Turismo)*, este último con el objetivo de usarse en las instalaciones hoteleras.

Todos tienen algo en común, y es que la mayoría de estos sistemas están especializados para el funcionamiento de determinadas empresas, de aquí que no sean flexibles, sino más bien, cerrados en su diseño, difíciles de estandarizar el proceso de gestión de inventarios y la integración con otros sistemas, elementos primordiales que se proponen resolver.

Razón ésta, por la que se hace preciso el desarrollo de un nuevo sistema de gestión de inventarios que cumpla con las necesidades actuales de las empresas y del país como tal, dotado de la tecnología más reciente, de modo que sea más flexible, moderno, amigable, confiable y seguro, y además que sea capaz de integrarse con otros sistemas. Precisando de este que sea configurable a partir de las necesidades particulares de los clientes, que cumpla con las nuevas regulaciones que establece el Ministerio de Finanzas y Precios. En fin, que cumpla con todos los nuevos requerimientos que los antiguos sistemas no brindan por haber quedado obsoletos.

Para el desarrollo satisfactorio de cualquier sistema de software y que a la vez el mismo garantice una determinada calidad y bajos costes, se hace necesario la utilización de metodologías de desarrollo, las que guían este proceso. Un elemento significativo en el proceso de desarrollo es la concepción de un diseño de alto nivel, que permita organizar y comprender la estructura del sistema que se desarrolla y brinde a todo el equipo una idea clara de lo que se está desarrollando.

Es un hecho que durante el proceso de desarrollo de software surgen situaciones que limitan y dificultan el proceso en sí. Entre ellos se pueden ver: la escasa reutilización, la mala selección de elementos de software, estilos arquitectónicos, patrones de arquitectura, protocolos de comunicación,

composición de elementos de diseño, asignación de funcionalidad a elementos del diseño, herramientas, frameworks, problemas de sincronización y acceso a datos, programadores escribiendo código de cualquier manera siempre y cuando cumpla con lo que quieren que haga el programa, influyendo todo esto en cuestiones de escalabilidad, rendimiento y costo. Donde la solución a todos estos problemas está en la correcta definición y descripción de una Arquitectura de Software.

La Arquitectura de Software surge de la necesidad de hacer los sistemas más modulares, que permitan la reutilización de componentes, por lo que su implementación reduce considerablemente el coste. Permite además mantener un bajo acoplamiento entre los elementos del sistema, pero con muy alta cohesión, ya que se establece bien claro la estructura de los componentes, sus formas de comunicarse y las relaciones que existen entre ellos.

Todo lo expuesto conllevó a la definición del siguiente **problema**:

¿Cómo tomar decisiones de diseño de alto nivel en el proceso de desarrollo del Sistema de Gestión de Inventarios que le permita a este, ser escalable, seguro, portable e integrable con otros sistemas de gestión empresarial?

La investigación se centra en el objeto de estudio: Proceso de Desarrollo del Sistema de Gestión de Inventario. Para ello se plantea el siguiente **objetivo general**: Proponer una Arquitectura de Software Orientada a Servicios que le permita al Sistema de Gestión de Inventario ser escalable, seguro, portable e integrable con otros sistemas de gestión empresarial.

Definiendo como **campo de acción**: Arquitectura de Software para el Sistema de Gestión de Inventario.

La investigación se realizó mediante las siguientes **tareas** para dar cumplimiento al objetivo propuesto:

- ◆ Realización de un estudio del estado del arte de la Arquitectura del Software para conocer los distintos enfoques de arquitectura, cómo evolucionaron, principales tendencias y definiciones, así como estilos arquitectónicos más significativos que están teniendo auge en el desarrollo de sistemas empresariales.
- ◆ Identificación de patrones y estilos arquitectónicos a utilizar, fundamentar la utilización de estos, así como su aplicación.
- ◆ Definición de los marcos de trabajo (frameworks) y las herramientas de desarrollo a utilizar.
- ◆ Definición y descripción de la Arquitectura.
- ◆ Validación por criterio de especialistas.

Para realizar las tareas se emplearon los siguientes **métodos**:

### **Métodos Teóricos**

- ◆ **Histórico – Lógico:** Para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.
- ◆ **Analítico – Sintético:** Para llegar a conclusiones en la investigación a partir de la información que se procese y precisar características del modelo arquitectónico propuesto.
- ◆ **Método Sistémico:** Para definir los elementos del sistema y sus relaciones.

### **Métodos Empíricos**

- ◆ **Medición:** para la evaluación de los tiempos de respuesta en situaciones críticas del caso de estudio.

### **Posibles resultados**

Propuesta de Arquitectura de Software Orientada a Servicios para el Sistema de Gestión de Inventarios que le permita a este, ser escalable, seguro, portable e integrable con otros sistemas de gestión empresarial.

### **El presente trabajo está estructurado en tres capítulos**

#### Capítulo 1: Fundamentación Teórica.

En este capítulo se realiza la fundamentación teórica de la investigación basándose en un estudio del estado del arte de la Arquitectura de Software, así como de los distintos enfoques de arquitectura, principales tendencias y definiciones. Se realiza un estudio de los estilos arquitectónicos más significativos que están teniendo auge, exponiendo sus ventajas y desventajas. En fin todos aquellos aspectos que permitirán modelar el marco teórico que fundamenta la investigación.

#### Capítulo 2:

En este capítulo se realiza la propuesta de solución al problema planteado en el diseño teórico de la investigación, la misma viene dada por el artefacto Documento Descripción de Arquitectura que propone la Universidad de las Ciencias Informáticas a emplear en los proyectos productivos.

#### Capítulo 3:

En este capítulo se realiza la especificación de las herramientas que brindarán soporte a la propuesta arquitectónica realizada en el capítulo anterior para el Sistema de Gestión de Inventario. Además se efectúa un análisis de la propuesta de solución desde el punto de vista de la calidad.

## CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

### 1.1. Introducción

En este capítulo se realiza la fundamentación teórica de la investigación, un estudio del arte de la Arquitectura de Software, así como de los distintos enfoques de arquitectura, principales tendencias y definiciones. Un estudio de los estilos arquitectónicos más significativos que están teniendo auge en el desarrollo de sistemas empresariales y de gestión, exponiendo sus ventajas y desventajas para seleccionar el adecuado para el desarrollo del Sistema de Gestión de Inventarios. Un análisis de los distintos niveles de abstracción de la Arquitectura de Software, los lenguajes de modelado arquitectónico, el estilo Arquitectura Orientada a Servicio como desafío del presente y la calidad en la Arquitectura de software.

### 1.2. Estado del arte de la Arquitectura de Software

La Arquitectura de Software es una práctica joven de apenas unos pocos años de trabajo constante. Tiene sus orígenes en la década del 60, pero no fue hasta los años 90 que Dewayne Perry y Alexander Wolf, la exponen en el sentido que hoy se conoce.

*“La década de 1990, creemos, será la década de la arquitectura de software. Usamos el término “arquitectura” en contraste con “diseño”, para evocar nociones de codificación, de abstracción, de estándares, de entrenamiento formal (de los arquitectos de software) y de estilo. Es tiempo de re-examinar el papel de la arquitectura de software en el contexto más amplio del proceso de software y de su administración, así como señalar las nuevas técnicas que han sido adoptadas.” (Perry, et al., 1992)*

Esta década se consideró como la década de la arquitectura de software, según profetizaron Perry y Wolf. A partir de este momento la Arquitectura de Software comenzó a tener auge vertiginoso tanto en la academia como en la industria.

#### 1.2.1. Principales definiciones

##### 1.2.1.1. Arquitectura de Software

La Arquitectura de Software es una disciplina reciente (Szyperski, 2000), pero no por ser una normativa joven deja de tener la importancia que requiere. A medida que crece la complejidad de las aplicaciones, y que se extiende el uso de sistemas distribuidos, los aspectos arquitectónicos del

desarrollo de software están recibiendo un interés cada vez mayor, tanto desde la comunidad científica como desde la propia industria del software (Velasco, 2000).

En la actualidad existen un sin número de definiciones de Arquitectura de Software, pero ninguna que sea adoptada completamente por la totalidad de los arquitectos del mundo. Aunque es válido aclarar que ninguna definición reprocha a la otra, sino que es otro modo de ver la cosas.

Entre las definiciones más reconocidas de la academia se encuentra la de Paul Clements: *“La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones”* (Reynoso, 2006).

De manera similar Bass define de la siguiente forma: *“La Arquitectura de Software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.”* (Pressman, 2002).

Por otra parte la definición que más reconocida internacionalmente por muchos arquitectos tributa a la industria y pertenece a la IEEE 1471, la cual cita así: *“La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.”* (Reynoso, 2006).

A pesar de la abundante cantidad de definiciones que existen de Arquitectura de Software, la gran mayoría coincide en que esta se refiere a la estructura a grandes rasgos del sistema (alto nivel de abstracción), estructura consistente en componentes y relaciones entre ellos. Por lo que se define Arquitectura de Software como la estructura y organización de los elementos de software de un Sistema Informático, cómo y bajo qué condiciones y configuraciones se van a comunicar dichos elementos.

Se puede concluir que la Arquitectura de Software es: una disciplina independiente de cualquier metodología de desarrollo, representa un alto nivel de abstracción del sistema y responde a los requerimientos no funcionales ya que estos se satisfacen mediante el modelado y diseño de la aplicación. De la misma forma se puede llegar a la conclusión de que la Arquitectura de Software no

es: diseño de software, una normativa madura, tratada de igual manera en la academia que en la industria.

### 1.2.1.2. Estilo

En general cuando se habla de la arquitectura, de una forma o de otra se refiere a algo en específico muy relacionado: las clasificaciones. El hecho es que no es solo como una cuestión clasificatoria, sino que cada forma arquitectónica tiene gran relación y se asocia con herramientas, conceptos, experiencias y problemas.

A estas clasificaciones se les ha llamado de varias formas, como: clases de arquitectura, tipos arquitectónicos, arquetipos recurrentes, especies, paradigmas topológicos, marcos comunes y varias docenas más. Desde hace más de una década esas cualificaciones arquitectónicas se vienen denominando: estilos, o alternativamente patrones de arquitectura. Los estilos sólo se manifiestan en arquitectura teórica descriptiva de alto nivel de abstracción; los patrones, por todas partes. (Reynoso, et al., 2004)

Las primeras definiciones explícitas de estilo parecen haber sido propuestas por *Dewayne Perry* y *Alexander Wolf* en octubre de 1992. Los mismos establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina. “*Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales*” (Perry, et al., 1992).

Según Pressman (Pressman, 2002), un estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen como se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema. El estilo arquitectónico es también un patrón de construcción.

Los estilos conjugan componentes, conectores, configuraciones y restricciones (constraints). La descripción de un estilo se puede formular en lenguaje natural o en diagramas, pero lo mejor es hacerlo en un *Lenguaje de Descripción Arquitectónica (ADL)* o en *Lenguajes Formales de Especificación (LFE)*.

Se puede concluir que:



- ◆ Un estilo contempla las cuatro “C” de la AS (elementos, conectores, configuraciones y restricciones), por sus siglas en inglés.
- ◆ Sirve para sintetizar estructuras de soluciones que luego serán refinadas a través del diseño.
- ◆ Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.
- ◆ Definen los patrones posibles de las aplicaciones, evitando errores arquitectónicos.
- ◆ Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

### 1.2.1.3. Patrón

*“Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puedes usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces”* (Fowler, et al., 2002).

Se podría ver a un patrón como una solución a un problema determinado, una codificación de algo específico basado en un conocimiento específico acumulado por la experiencia.

*“Un sistema bien estructurado esta lleno de patrones”* (Reynoso, 2005).

Se puede decir que cada vez que se utiliza un patrón se utiliza un poco de aquí y otro poco de allá. Se verá la misma solución varias veces, pero nunca exactamente igual a ninguna otra.

Si se hace una primera clasificación de los patrones software atendiendo al nivel de abstracción de éstos, pudieran quedar: los patrones arquitectónicos, de diseño o centrados en el código, entre otros.

**Elementos de un Patrón** (Reynoso, 2005):

- ◆ Nombre
  - Define un vocabulario de diseño
  - Facilita abstracción
- ◆ Problema
  - Describe cuando aplicar el patrón
  - Conjunto de fuerzas: objetivos y restricciones
  - Prerrequisitos
- ◆ Solución
  - Elementos que constituyen el diseño (template)
  - Forma canónica para resolver fuerzas
- ◆ Consecuencias

- Resultados, extensiones y tradeoffs

### **Patrón Arquitectónico**

Según Buschmann los patrones de arquitectura se pueden ver como la descripción de un problema en particular y recurrente de diseño, que aparece en contextos de diseño arquitectónicos específicos, y representa un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma en que estos colaboran entre sí. (Buschmann, et al., 1996)

Los patrones arquitectónicos se definen sobre aspectos fundamentales de la estructura del sistema software, donde se especifican un conjunto de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. Posteriormente, el diseño hará uso de patrones de diseño para resolver problemas específicos.

### **Patrón de Diseño**

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschmann, et al., 1996).

Son menores en la escala de abstracción que los patrones arquitectónicos, además tienden a ser independientes de los lenguajes y paradigmas de programación. Su aplicación tiene efectos sobre subsistemas, debido a que especifica a un mayor nivel de detalle, sin llegar a la implementación, el comportamiento de los componentes del subsistema.

#### **1.2.1.4. Marco de trabajo**

El concepto de marco de trabajo (*framework*) no es sencillo de definir, a pesar de que cualquiera con experiencia programando captará su sentido de manera casi intuitiva, y es muy posible que esté utilizando su propio *framework*.

El concepto *framework* se emplea en muchos ámbitos del desarrollo de sistemas software. Se pueden encontrar *frameworks* para el desarrollo de aplicaciones web, aplicaciones de escritorio, aplicaciones médicas, para el desarrollo de juegos, y para cualquier ámbito que se le pueda ocurrir a uno.

En general, el término *framework*, se refiere a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un

framework se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede añadir las últimas piezas para desarrollar una aplicación concreta.

Por lo que resumiendo, un framework es el esqueleto de una aplicación que debe ser adaptado por el programador según sus necesidades específicas para desarrollar una aplicación.

### 1.2.2. Tendencias

Aunque no existe hoy en día nada que avale explícitamente la existencia de escuelas de Arquitectura de Software, ni una referencia que analice las particularidades de cada una, en la actualidad se pueden distinguir a grandes rasgos unas seis corrientes. El pertenecer a una de estas corrientes no representa algo determinista, ya que en distintos momentos algunos practicantes de la Arquitectura de Software realizan diferentes trabajos operando en marcos distintos, o porque simplemente cambian de concepción.

La división de escuelas de Arquitectura de Software que se propone es la siguiente:

1. **Arquitectura como etapa de ingeniería y diseño orientada a objetos.** Evidentemente sería la de Grady Booch; para él, la Arquitectura de Software es *“la estructura lógica y física de un sistema, forjada por todas las decisiones estratégicas y tácticas que se aplican durante el desarrollo”* [Boo91]. Otras definiciones revelan que la Arquitectura de Software, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descritas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten. (Reynoso, 2006)
2. **Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.** Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon. Se trata también de la visión de la Arquitectura de Software dominante en la academia, y aunque es la que ha hecho el esfuerzo más importante por el reconocimiento de la Arquitectura de Software como disciplina, sus categorías y herramientas son todavía mal conocidas en la práctica industrial. En el interior del movimiento se pueden observar distintas divisiones. Hay una variante informal de “boxología” y una vertiente más formalista, representada por el grupo de Mark Moriconi en el SRI de Menlo Park. En principio se pueden reconocer tres modalidades en cuanto a la formalización; los más informales utilizan descripciones verbales o diagramas de cajas, los de talante intermedio se sirven de ADLs y los más exigentes usan lenguajes formales de especificación como CHAM y Z. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración

explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código, y en general nadie habla de clases o de objetos. Mientras algunos participantes excluyen el modelo de datos de las incumbencias de la Arquitectura de Software (Shaw, Garlan, etc), otros insisten en su relevancia (Medvidovic, Taylor). (Reynoso, 2006)

3. **Estructuralismo arquitectónico radical.** Se trata de un desprendimiento de la corriente anterior, mayoritariamente europeo, que asume una actitud más confrontativa con el mundo UML. En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos para la Arquitectura de Software y otra que procura definir nuevos meta-modelos y estereotipos de UML como correctivos de la situación. (Reynoso, 2006)
4. **Arquitectura basada en patrones.** Si bien reconoce la importancia de un modelo emanado del diseño orientado a objetos, no se encuentra tan rígidamente vinculada a UML en el modelado, ni a CMM en la metodología. El texto sobre patrones que esta variante reconoce como referencia es la serie POSA de Buschmann y otros, y secundariamente el texto de la Banda de los Cuatro [Gof95]. El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura. (Reynoso, 2006)
5. **Arquitectura procesual.** Desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos de los arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, entre otros. Intenta establecer modelos de ciclo de vida y técnicas de elicitación de requerimientos, brainstorming, diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la Arquitectura de Software. (Reynoso, 2006)
6. **Arquitectura basada en escenarios.** Es la corriente más nueva. Se trata de un movimiento predominantemente europeo, con centro en Holanda. Recupera el nexo de la Arquitectura de Software con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. En esta corriente suele utilizarse diagramas de casos de uso UML como herramienta informal u ocasional, dado que los casos de uso son uno de los escenarios posibles. Los casos de uso no están orientados a objeto. Los autores vinculados con esta modalidad han sido, aparte de los codificadores de ATAM, CBAM, QASAR y demás métodos del SEI, los arquitectos holandeses de la Universidad Técnica de Eindhoven, de la Universidad Brije, de la Universidad de Groningen y de Philips Research. (Reynoso, 2006)

### 1.2.3. Diferencia entre Arquitectura de Software y Diseño

La comunidad académica de Arquitectura de Software difiere sustancialmente de confundir diseño con arquitectura, aunque algunos practicantes a la hora de referirse al tema dejan muy ambiguo la relación que existe entre ambos campos.

En alguna medida, la arquitectura y el diseño sirven al mismo propósito. Sin embargo, la Arquitectura de Software se centra en la estructura del sistema y en las interconexiones de los componentes, distinguiéndose del diseño de software tradicional, tales como el diseño orientado a objetos, que se concentra más en el modelado de abstracciones de más bajo nivel, tales como algoritmos y tipos de datos. A medida que la arquitectura de alto nivel se refina, sus conectores pueden perder prominencia, distribuyéndose a través de los elementos arquitectónicos de más bajo nivel, resultando en la transformación de la arquitectura en diseño. (Reynoso, et al., 2004).

La Arquitectura de Software, es una disciplina relativamente nueva en materia de diseño de software y en realidad abarca también las metodologías de diseño, así como metodologías de análisis. El arquitecto de software contemporáneo, ejecuta una combinación de roles como los de analista de sistemas, diseñador de sistemas e ingeniero de software. Pero la arquitectura es más que una relocalización de funciones. El concepto de arquitectura intenta subsumir las actividades de análisis y diseño en un framework de diseño más amplio y más coherente. Pero la arquitectura es algo más integrado que la suma del análisis por un lado y el diseño por el otro. La integración de metodologías y modelos, es lo que distingue la AS de la simple yuxtaposición de técnicas de análisis y de diseño. (Albin, 2003).

La arquitectura concierne a un nivel de abstracción más elevado; se ocupa de componentes y no de procedimientos; de las interacciones entre esos componentes y no de las interfaces; de las restricciones a ejercer sobre los componentes y las interacciones y no de los algoritmos, los procedimientos y los tipos. En cuanto a la composición, la de la arquitectura es de grano grueso, la del diseño es de fina composición procedural; las interacciones entre componentes en arquitectura tienen que ver con un protocolo de alto nivel (en el sentido no técnico de la palabra), mientras que las del diseño conciernen a interacciones de tipo procedural (mensajes, llamadas a rutinas) (Perry, 1997).

Por lo que se puede concluir que la Arquitectura de Software no es diseño de software, ya que ocurre en una etapa más temprana del desarrollo de un proyecto a un alto nivel de abstracción ocupándose de componentes, conectores, configuraciones, restricciones, dejándole al diseño detallar elementos arquitectónicos de más bajo nivel, más refinados como: procedimientos, algoritmos y tipos de datos.

## 1.3. Niveles de Abstracción de la Arquitectura de Software

### 1.3.1. Estilos Arquitectónicos

#### 1.3.1.1. Clasificación de Estilos

##### ¿Cuántos y cuáles son los estilos?

En un estudio realizado por Mary Shaw y David Garlan (Garlan, et al., 1994), proponen la siguiente taxonomía, en la que se entremezclan lo que antes se llamaba “arquitecturas” con los “modelos de diseño”:

1. Tubería-filtros
2. Organización de abstracción de datos y orientación a objetos
3. Invocación implícita, basada en eventos
4. Sistemas en capas
5. Repositorios
6. Intérpretes orientados por tablas
7. Procesos distribuidos. Una forma particular de proceso distribuido es, por ejemplo, la arquitectura cliente-servidor.
8. Organizaciones programa principal / subrutina.
9. Arquitecturas de software específicas de dominio
10. Sistemas de transición de estado
11. Sistemas de procesos de control
12. Estilos heterogéneos

En *Software Architecture in Practice*, un texto fundamental de *Bass, Clements y Kazman* (Bass, et al., 1998), se proporciona una sistematización de clases de estilo en cinco grupos:

- ◆ **Flujo de datos** (movimiento de datos, sin control del receptor de lo que viene “corriente arriba”)
  - Proceso secuencial por lotes
  - Red de flujo de datos
  - Tubería-filtros
- ◆ **Llamado y retorno** (estilo dominado por orden de computación, usualmente con un solo thread de control)
  - Programa principal / Subrutinas
  - Tipos de dato abstracto
  - Objetos

- Cliente-servidor basado en llamadas
- Sistemas en capas
- ◆ **Componentes independientes** (dominado por patrones de comunicación entre procesos independientes, casi siempre concurrentes)
  - Sistemas orientados por eventos
  - Procesos de comunicación
- ◆ **Centrados en datos** (dominado por un almacenamiento central complejo, manipulado por computaciones independientes)
  - Repositorio
  - Pizarra
- ◆ **Máquina virtual** (caracterizado por la traducción de una instrucción en alguna otra)
  - Intérprete

Los estilos a diferencia de patrones son unos pocos, y se agrupan en clases de estilos, siendo la clasificación más actual, concisa y que se asume durante la investigación la siguiente (Reynoso, 2005):

- ◆ **Estilos de Flujo de Datos**
  - Tubería y filtros
- ◆ **Estilos Centrados en Datos**
  - Arquitecturas de Pizarra o Repositorio
- ◆ **Estilos de Llamada y Retorno**
  - Model-View-Controller (MVC)
  - Arquitecturas en Capas
  - Arquitecturas Orientadas a Objetos
  - Arquitecturas Basadas en Componentes
- ◆ **Estilos de Código Móvil**
  - Arquitectura de Máquinas Virtuales
- ◆ **Estilos heterogéneos**
  - Sistemas de control de procesos
  - Arquitecturas Basadas en Atributos
- ◆ **Estilos Peer-to-Peer**
  - Arquitecturas Basadas en Eventos
  - Arquitecturas Orientadas a Servicios (SOA)
  - Arquitecturas Basadas en Recursos

### 1.3.1.2. Estilos más Populares

En la década de 1960 y 1970 era natural estructurar una aplicación en términos de los servicios funcionales que ésta debía proveer. De hecho, las metodologías como el análisis y diseño estructurado partían de una descomposición funcional jerárquica que luego se transformaba en procedimientos dentro de la aplicación. Con el aumento de la complejidad de las aplicaciones y la necesidad de hacerlas evolucionar, este enfoque no fue suficiente. En los 80s y 90s se popularizó el desarrollo de software orientado-objetos que buscaba, entre otros objetivos, permitir la construcción de aplicaciones más mantenibles. (Casallas, et al., 2005)

Hoy parecen no ser suficiente los objetos, no porque hayan dejado de ser buenos, sino porque el problema ha cambiado. Estos cambios son consecuencia de los avances tecnológicos y el incremento de las expectativas de los clientes. Por ejemplo, las características no funcionales se han vuelto más críticas y el problema de la integración de aplicaciones ha llegado a ser prioritario. Se necesita contar con arquitecturas flexibles y lo más independientes posibles de las herramientas que se utilicen. En el (anexo 1) se muestra como han ido evolucionando las arquitecturas de los sistemas a lo largo de estos años.

#### ❖ Arquitectura Orientada a Objetos

A este tipo de arquitectura se le conoce de varias maneras, entre ellas: Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos.

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. (Garlan, et al., 1994)

Según Billy Reinoso (Reynoso, et al., 2004), si hubiera que resumir las características de las arquitecturas orientadas a objetos, se podría decir que:

- ◆ Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.



- ◆ Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología apenas importa si los objetos son locales o remotos. El mejor ejemplo de orientación a objetos para sistemas distribuidos es *Common Object Request Broker Architecture* (CORBA), en la cual las interfaces se definen mediante *Interface Description Language* (IDL); un *Object Request Broker* media las interacciones entre objetos clientes y objetos servidores en ambientes distribuidos.
- ◆ En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. Hay muchas variantes del estilo; algunos sistemas, por ejemplo, admiten que los objetos sean tareas concurrentes; otros permiten que los objetos posean múltiples interfaces.

**Ventajas:**

- ◆ Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- ◆ Un objeto es una entidad reutilizable en el entorno de desarrollo
- ◆ Encapsulamiento.

**Desventajas:**

- ◆ Para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.
- ◆ Presenta problemas de efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar a A.
- ◆ Granularidad muy fina para sistemas grandes.

**❖ Arquitectura en Capas**

Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior (Reynoso, et al., 2004). (Ver anexo 2)

En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción. Las restricciones topológicas del estilo pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes, y a los elementos de una capa entenderse sólo con otros elementos de la misma; se supone que si esta exigencia se relaja, el estilo deja de ser puro y pierde algo de su capacidad heurística (MSDN, 2004); también se pierde,

naturalmente, la posibilidad de reemplazar de cuajo una capa sin afectar a las restantes, disminuye la flexibilidad del conjunto y se complica su mantenimiento.

A veces se argumenta que el cruce excesivo de muchos niveles involucra eventuales degradaciones de performance; pero muchas más veces se sacrifica la pureza de la arquitectura en capas precisamente para mejorarla: colocando, por ejemplo, reglas de negocios en los procedimientos almacenados de las bases de datos, o articulando instrucciones de consulta en la capa de la interface del usuario.

### **Ventajas**

- ◆ Modularidad (hasta cierto punto)
- ◆ Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- ◆ Proporciona amplia reutilización.
- ◆ Soporta fácilmente la evolución del sistema; los cambios sólo afectan a las capas vecinas
- ◆ Se pueden cambiar las implementaciones respetando las interfaces con las capas adyacentes.

### **Desventajas**

- ◆ Es difícil razonar a priori sobre la separación en capas requerida
- ◆ Formatos, protocolos y transportes de la comunicación entre capas suelen ser específicos y propietarios
- ◆ No todos los sistemas pueden estructurarse en capas.

### **❖ Arquitectura basada en Componentes**

Actualmente en el desarrollo de software hay una gran necesidad de hacer uso de la reutilización de partes o módulos de software existente, que podrían ser utilizadas para la generación de nuevas extensiones de las aplicaciones o las aplicaciones completas. Cuando se hable de reutilización en los procesos de ingeniería, está muy implícito el concepto de “componente”, pues a las partes eficientes de software que pueden ser utilizadas para la construcción de aplicaciones se les conoce como “componentes software”.

Son varias las definiciones este término (Componente de Software):

- ◆ Según *Krutchen*, un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un

componente cumple con un conjunto de interfaces y provee la realización física de ellas. (Brown, et al., 1998).

- ◆ El *Software Engineering Institute* (SEI) de la Universidad Carnegie Mellon formuló una definición con el propósito de consolidar las diferentes opiniones acerca de lo que debía ser un componente de software: “*un componente es una implementación opaca de funcionalidad, sujeta a composición por terceros y que cumple con un modelo de componentes*” (Bachmann, et al., 2000).
- ◆ Según Szyperski, un componente de software es una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Un componente de software puede ser desplegado independientemente y está sujeto a composición por terceros. (Szyperski, 2002).

Se puede ver que la mayoría de las definiciones giran sobre lo esencial de un componente, que es una implementación de un artefacto de software, que ejecuta una o varias funcionalidades, liberadas por servicios, vistos como un conjunto de interfaces y provee la realización física de ellas.

### Características:

Una de las características más importantes de los componentes es que son reutilizables. Para ello los componentes deben satisfacer como mínimo el siguiente conjunto de características:

- ◆ **Identificable:** un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.
- ◆ **Accesible sólo a través de su interfaz:** el componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.
- ◆ **Servicios son invariantes:** las operaciones que ofrece un componente, a través de su interfaz, no deben variar. La implementación de estos servicios puede ser modificada, pero no deben afectar la interfaz.
- ◆ **Documentado:** un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte.

La evaluación dominante del estilo de componentes subraya su mayor versatilidad respecto del modelo de objetos, pero también su menor adaptabilidad comparado con el estilo orientado a servicios (Reynoso, et al., 2004).

**Ventajas:**

- ◆ **Reutilización del software.** Lleva a alcanzar un mayor nivel de reutilización de software.
- ◆ **Simplifica las pruebas.** Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- ◆ **Simplifica el mantenimiento del sistema.** Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- ◆ **Mayor calidad.** Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

**Desventajas:**

- ◆ Si no existen los componentes, hay que desarrollarlos y se puede perder mucho tiempo, así como en que estos componentes pueden tener conflictos si de estos sale una nueva versión es posible que haya que re-implementar estos componentes.
- ◆ Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

**❖ Arquitectura Orientada a Servicios (SOA)**

Las Arquitecturas Orientadas a Servicios (SOA: *Service Oriented Architecture*) están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma y del lenguaje de programación (por ejemplo WSDL: *Web Services Description Language*). La definición de la interfaz encapsula las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo (como *Java* o *.NET*).

Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio desarrollado en *CSharp* podría ser usado por una aplicación Java.

Una de las características más resaltante de SOA, es que esta basada en contratos, donde el proveedor establece las reglas de comunicación, el transporte, y los datos de entrada y salida que serán intercambiados por ambas partes. Para ver mas características de SOA ver (anexo 3).

### ¿Qué es SOA?

- ◆ SOA es una arquitectura conceptual.
- ◆ Organiza funciones de negocio como servicios interoperables.
- ◆ Permite reutilización de servicios para satisfacer necesidades de negocio.
- ◆ SOA es basado en estándares.
- ◆ Independencia de fabricantes.
- ◆ SOA es una estrategia de IT a nivel empresarial.

SOA es un estilo de arquitectura en el cual se exponen los procesos de negocio del sistema a construir como servicios independientes de alta cohesión y bajo acoplamiento que encapsulan dichos procesos y pueden ser invocados a través de interfaces bien definidas.

### ¿Qué no es SOA

SOA como tal no es:

- ◆ Metodología de Desarrollo de Proyectos.
- ◆ Metodología de Arquitectura Empresarial.

### SOA como estilo arquitectónico

- ◆ Componente: Servicio
- ◆ Conectores: Antes, RPC – Ahora, paso de mensajes.
- ◆ Configuración: Distribuido
- ◆ Restricciones (*Constraint*): Bajo acoplamiento, independencia de modelo de programación, independencia de plataforma, transporte y protocolo por acuerdo de industria

### Ventajas

- ◆ Mejorar toma de decisiones.
  - Panorámica unificada. Más información con mejor calidad.
- ◆ Mejorar productividad de empleados.
  - Acceso óptimo a sistemas. No limitación de TIC
- ◆ Potenciar relación con los clientes y proveedores.
  - Mayor capacidad de respuesta a los clientes.
- ◆ Aplicaciones más productivas y flexibles.
- ◆ Aplicaciones más seguras y manejables.

### Desventajas

- ◆ Los tiempos de llamado no son despreciables, gracias a la comunicación de la red, tamaño de los mensajes, entre otros. Esto necesariamente implica la utilización de mensajería confiable.
- ◆ La respuesta del servicio es afectada directamente por aspectos externos como problemas en la red, configuración, entre otros.
- ◆ Debe manejar comunicaciones no confiables, mensajes impredecibles, reintentos, mensajes fuera de secuencia, etcétera.

### 1.3.2. Patrones Arquitectónicos

Existe una categorización muy utilizada dividida en 2 de los sistemas de patrones de arquitectura más extendidos, siendo los mismos: el de *Pattern Oriented Software Architecture* (POSA) (Buschmann, et al., 1996) y el de *Pattern of Enterprise Application Architecture* (PEAA) (Fowler, et al., 2002).

#### Categorías de POSA

En el libro de referencia de patrones de arquitectura POSA, se divide a los patrones de la siguiente forma:

- ◆ **From Mud to Structure:** Estos patrones ayudan a evitar una abundancia de componentes u objetos. En particular, soportan una descomposición controlada de una tarea del sistema en sub-tareas que cooperan.
- ◆ **Distributed Systems**
- ◆ **Interactive Systems**
- ◆ **Adaptable Systems**

En el (anexo 4) se muestra la distribución de los patrones de POSA en las categorías enunciadas y seguidamente en el (anexo 5) se muestra una breve explicación de algunos de ellos.

#### Categorías de PEAA

En PEAA (Fowler, et al., 2002), describen una gran cantidad de patrones orientados a la arquitectura de aplicaciones empresariales.

En PEAA se definen las siguientes categorías de patrones:

- ◆ **Layering:** Patrones para dividir un sistema en capas.
- ◆ **Organización de la lógica del dominio:** Formas de organizar los objetos del dominio.

- ◆ **Mapping to Relational Databases:** Se relaciona con la comunicación entre la lógica del dominio y los repositorios de datos. Incluye el mapeo entre modelos de objetos y bases de datos relacionales.
- ◆ **Web Presentation:** La presentación Web es uno de los desafíos que han tenido que sortear en los últimos años las aplicaciones empresariales. Los clientes delgados Web proveen muchas ventajas, siendo una de las principales la facilidad de distribución (no es necesario instalar software en los equipos cliente). Esta categoría incluye una serie de patrones para gestionar la presentación Web. (Welicki, 2007)
- ◆ **Concurrency:** Manejo de la concurrencia. Las aplicaciones actuales basadas en tecnologías Web tienen grandes necesidades de gestión de la concurrencia.
- ◆ **Sesion State:** Patrones para el manejo de la sesión en servidores Web.
- ◆ **Distribution Strategies:** Distribución de objetos en múltiples emplazamientos, basada en conocimientos empíricos en clientes.

En el (anexo 6) se muestra la distribución de los patrones definidos en PEAA en las categorías antes expuestas.

Los términos *patrón de arquitectura* y *estilo de arquitectura* suelen confundirse y muchos de los estudiosos de la disciplina parecen no estar bien claros al respecto. Con la intención de hacer una comparación clara entre estilo arquitectónico y patrón arquitectónico, en (anexo 7) se presenta algunas de las diferencias entre estos conceptos, construida a partir del planteamiento (Buschmann, et al., 1996).

### 1.3.3. Relación entre los Niveles de Abstracción

En (anexo 8) se muestra la relación de abstracción existente entre los conceptos de estilo arquitectónico, patrón arquitectónico y patrón de diseño.

Los estilos y patrones ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requerimientos de calidad permitiendo de esa manera que el sistema de software a construir perdure en el tiempo.

Ahora bien, la organización del sistema de software debe estar disponible para todos los involucrados en el desarrollo del sistema, ya que establece un mecanismo de comunicación entre los mismos. Esto se logra mediante la representación de la arquitectura en formas distintas, obteniendo así una descripción de la misma de forma tal que puede ser entendida y analizada por todos los involucrados,

con miras a obtener un sistema de calidad. Estas descripciones pueden establecerse a través de las vistas arquitectónicas, las notaciones como UML y los lenguajes de descripción arquitectónica (Bengtsson, 1999).

Las vistas arquitectónicas, a través de distintos niveles de abstracción, resaltan varias cuestiones que conciernen a los involucrados en el desarrollo. Por lo que resulta interesante analizar estas perspectivas de una arquitectura, y la forma en que éstas ayudan a todos los involucrados en el proceso de desarrollo a razonar sobre los atributos de calidad del sistema.

#### 1.3.4. Marcos de Trabajo

En la sección 1.2.1.4 se ha dado una breve definición sobre lo que es un framework por lo que esta sección se intentará abordar más sobre los mismos.

Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones, entre otros.

La labor del usuario del framework debe ser, por un lado, de seleccionar, instanciar, extender y reutilizar los componentes que este proporciona, y por otro, completar la arquitectura del mismo desarrollando componentes específicos, que deben ser acoplados en el framework, logrando así desarrollar diferentes aplicaciones siguiendo las restricciones estructurales impuestas por el framework.

#### ¿Qué ventajas tiene utilizar un framework?

Las que se derivan de utilizar un estándar; entre otras:

- ◆ El programador no necesita plantearse una estructura global de la aplicación, sino que el framework le proporciona un esqueleto que hay que “rellenar”.
- ◆ Facilita la colaboración. Cualquiera que haya tenido que *pelearse* con el código fuente de otro programador, o incluso con el propio, sabrá lo difícil que es entenderlo y modificarlo; por tanto, todo lo que sea definir y estandarizar va a ahorrar tiempo y trabajo a los desarrollos colaborativos.
- ◆ Es más fácil encontrar herramientas (utilidades, librerías) adaptadas al framework concreto para facilitar el desarrollo.

Aunque el uso de los frameworks sea una buena práctica para el desarrollo de un sistema software no significa que haya que utilizar obligatoriamente uno, ya sea conocido o no, ni nada por el estilo. Esto va a estar en dependencia de la magnitud del proyecto y de la experiencia que se tenga por parte del equipo de desarrollo y el arquitecto.



Según uno de los principios de la programación segura, ¿Por qué no utilizar un framework ya definido, y aprovechar el trabajo de otros desarrolladores?, o sea, ¿Por qué reinventar la rueda? Si se tiene algo que probado, que funciona, y está en explotación, no hay necesidad de hacer algo que haga lo mismo, esto es poco útil y solo puede traer complicaciones como la pérdida de tiempo.

Es cierto que la utilización de un framework implica cierto coste inicial (curva de aprendizaje), pero esto es algo que se compensa a la larga durante el desarrollo de un proyecto software.

## 1.4. SOA: Desafío del presente

La Arquitectura Orientada a Servicios es un concepto de arquitectura de software que define la utilización de servicios como construcciones básicas para el desarrollo de aplicaciones. Es una arquitectura de una aplicación donde las funcionalidades se definen como servicios independientes, con interfaces accesibles, bien definidas, que pueden ser llamadas en secuencias dadas para formar procesos de negocios.

*“SOA ha surgido como la mejor manera de afrontar el desafío de hacer más con menos recursos. Promete hacer la reutilización y la integración mucho más fáciles, ayudando a reducir el tiempo de desarrollo y aumentando la agilidad organizacional. No sorprendentemente, el 80% de las organizaciones de IT están implementando aplicaciones usando SOA con web services subyacentes. SOA proporciona mayor flexibilidad para afrontar los cambios tanto en el ambiente de negocios como en la infraestructura tecnológica” (Reynoso, 2005).*

Gartner, proveedor líder de investigación y análisis de la industria global de tecnologías de información (TI), en el 2003 plantea: *“Hacia 2008, SOA será la práctica prevalente de ingeniería de software, acabando con los 40 años de dominación de las arquitecturas monolíticas (probabilidad 0.7)”* (Natis, 2003).

### 1.4.1. Razones para usar SOA

Existen varias razones para que una empresa adopte un enfoque SOA, y más concretamente un enfoque SOA basado en servicios web:

- ◆ **Reutilización:** El factor fundamental en el cambio a SOA es la reutilización de los servicios de negocio. Las funciones de negocio, dentro de una empresa y con los business partners<sup>1</sup>,

---

<sup>1</sup> **Business partners:** Persona o empresa que es copropietaria de la empresa o compañía, en otras palabras, es un socio de la compañía al que le pertenece parte de ella.

pueden ser expuestos como servicios web y ser reutilizadas para cubrir nuevas necesidades de negocio.

- ◆ **Interoperabilidad:** El objetivo de una arquitectura débilmente acoplada es que los clientes y servicios se comuniquen independientemente de la plataforma en que residan. Los protocolos de comunicación con servicios web son independientes de la plataforma, lenguaje de codificación y sistema operativo por lo que facilitan la comunicación con los business partners.
- ◆ **Escalabilidad:** Como los servicios de SOA están débilmente acoplados, las aplicaciones que usan esos servicios escalan fácilmente. Esto es debido a que existe muy poca dependencia entre las aplicaciones clientes y los servicios que usan.
- ◆ **Flexibilidad:** Es otra de las características que proporciona el acoplamiento débil entre los servicios. Cualquier cambio en la implementación de uno de ellos no afectaría al resto siempre que se mantenga la interfaz.
- ◆ **Eficiencia de coste:** Las arquitecturas SOA se basan en la exposición de servicios ya existentes para ser reutilizados. Al usar servicios web, para exponer estos servicios, se reutilizan la infraestructura web existente en virtualmente todas las organizaciones por lo que se limita considerablemente el coste.

### 1.4.2. Elementos de SOA

Esta arquitectura presenta una forma de construir sistemas distribuidos que entreguen a la aplicación funcionalidad como servicios para aplicaciones de uso final u otros servicios.

En el anexo 9 se muestran los elementos que podrían observarse en una arquitectura orientada a servicios. En el mismo se pueden diferenciar dos zonas, una que abarca los aspectos funcionales de la arquitectura y otra que abarca aspectos de calidad de servicio. A continuación se describen los elementos brevemente (Endrei, et al., 2004):

#### ◆ **Funciones**

- **Transporte:** es el mecanismo utilizado para llevar las demandas de servicio desde un consumidor de servicio hacia un proveedor de servicio, y las respuestas desde el proveedor hacia el consumidor.
- **Protocolo de comunicación de servicios:** es un mecanismo acordado a través del cual un proveedor de servicios y un consumidor de servicios comunican qué está siendo solicitado y qué está siendo respondido.
- **Descripción de servicio:** es un esquema acordado para describir qué es el servicio, cómo debe invocarse, y qué datos requiere el servicio para invocarse con éxito.

- **Servicios:** describe un servicio actual que está disponible para utilizar.
- **Procesos de Negocio:** es una colección de servicios, invocados en una secuencia particular con un conjunto particular de reglas, para satisfacer un requerimiento de negocio.
- **Registro de Servicios:** es un repositorio de descripciones de servicios y datos que pueden utilizar proveedores de servicios para publicar sus servicios, así como consumidores de servicios para descubrir o hallar servicios disponibles.
- ◆ **Calidad de Servicio**
  - **Política:** es un conjunto de condiciones o reglas bajo las cuales un proveedor de servicio hace el servicio disponible para consumidores.
  - **Seguridad:** es un conjunto de reglas que pueden aplicarse para la identificación, autorización y control de acceso a consumidores de servicios.
  - **Transacciones:** es el conjunto de atributos que podrían aplicarse a un grupo de servicios para entregar un resultado consistente.
  - **Administración:** es el conjunto de atributos que podrían aplicarse para manejar los servicios proporcionados o consumidos.

Las colaboraciones en SOA siguen el paradigma *publicar, descubrir e invocar*, donde un consumidor de servicios realiza la localización dinámica de un servicio consultando el registro de servicios para hallar uno que cumpla con un determinado criterio. Si el servicio existe, el registro proporciona al consumidor la interfaz de contrato y la dirección del servicio proveedor. (Endrei, et al., 2004)

En el anexo 10 se ilustra las entidades (roles, operaciones y artefactos) en una Arquitectura Orientada a Servicios donde estas colaboran.

Cada entidad de las que se muestra en el diagrama del (anexo 10) puede tomar el rol de consumidor, proveedor y/o registro:

- ◆ Un **consumidor de servicios** es una aplicación, un módulo de software u otro servicio que requiere un servicio, y ejecuta el servicio de acuerdo a un contrato de interfaz.
- ◆ Un **proveedor de servicios** es una entidad direccionable a través de la red que acepta y ejecuta consultas de consumidores, y publica sus servicios y su contrato de interfaces en el registro de servicios para que el consumidor de servicios pueda descubrir y acceder al servicio.
- ◆ Un **registro de servicios** es el encargado de hacer posible el descubrimiento de servicios, conteniendo un repositorio de servicios disponibles y permitiendo visualizar las interfaces de los proveedores de servicios a los consumidores interesados.

Las operaciones son:

- ◆ **Publicar.** Para poder acceder a un servicio se debe publicar su descripción para que un consumidor pueda descubrirlo e invocarlo.
- ◆ **Descubrir.** Un consumidor de servicios localiza un servicio que cumpla con un cierto criterio consultando el registro de servicios.
- ◆ **Ligar e Invocar.** Una vez obtenida la descripción de un servicio por parte de un consumidor, éste lo invoca de acuerdo a la información en la descripción del servicio.

Finalmente, los artefactos en una arquitectura orientada a servicios son (Alvez, et al., 2006):

- ◆ **Servicio.** Un servicio que está disponible para el uso a través de una interfaz publicada y que permite ser invocado por un consumidor de servicios.
- ◆ **Descripción de servicio.** Una descripción de servicio especifica la forma en que un consumidor de servicio interactuará con el proveedor de servicio, especificando el formato de consultas y respuestas desde el servicio. Esta descripción también puede especificar el conjunto de precondiciones, pos-condiciones y/o niveles de calidad de servicio.

Analizando lo anteriormente planteado se puede concluir que el elemento básico en este tipo de arquitectura es el servicio, pero únicamente con este artefacto no se podría diseñar una SOA. Para conformar una SOA se necesita básicamente la conjunción de operaciones, servicios, mensajes y procesos de negocio.

#### 1.4.2.1. Tipos de Servicios

- ◆ **Servicios básicos:** pueden estar centrados en datos o en lógica y encapsulan funcionalidades como cálculos complejos, acceso a datos y reglas complejas de negocio.
- ◆ **Servicios intermediarios:** servicios adaptadores, fachadas, etcétera. Suelen ser servicios sin estado.
- ◆ **Servicios de proceso:** servicios de negocio que encapsulan la lógica de proceso. Suelen conservar estado y pueden residir en herramientas BPM.
- ◆ **Servicios públicos:** servicios accesibles por terceros (fuera de la organización).

Un **servicio de negocio** es un componente reutilizable de software, con significado funcional completo, y que está compuesto por (ver anexo 11):

- ◆ **Contrato:** especificación de la finalidad, funcionalidad, forma de uso y restricciones del servicio.
- ◆ **Interfaz:** mecanismo de exposición del servicio a los usuarios.
- ◆ **Implementación:** debe contener la lógica o el acceso a datos.

### 1.4.3. Servicios Web

Las Arquitecturas Orientadas a Servicios, en contraste con las Arquitecturas Orientadas a Objetos, están formadas por servicios de aplicación altamente interoperables y débilmente acoplados. Donde estos servicios se pueden ver como la evolución en complejidad de un componente distribuido, diferenciándose en que son (Alvez, et al., 2006):

- ◆ Mucho menos acoplados con sus aplicaciones cliente que los componentes.
- ◆ Menor granularidad que los componentes.
- ◆ No son diseñados e implementados necesariamente como parte de una aplicación end-to-end.
- ◆ Son controlados y administrados de manera independiente.
- ◆ Expone su funcionalidad a través de protocolos abiertos e independientes de plataforma. Incluso arriesgando el rendimiento y consumo de recursos.
- ◆ Son transparentes de su localización en la red, de esta manera garantizan escalabilidad y tolerancia a fallos.

Por lo general, los servicios incluyen tanto lógica de negocios como manejo de datos, relevantes a la solución del problema para el cual fueron diseñados. Un servicio funciona como una aplicación independiente, teniendo sus propias reglas de negocio, datos, procedimientos de administración y operación, políticas de escalabilidad, seguridad, tolerancia a fallos, manejo de excepciones y configuración. Expone toda su funcionalidad utilizando una interfaz basada en mensajes, por lo que la comunicación con el servicio, es realizada mediante mensajes y no llamadas a métodos. Estos mensajes deben contener o referenciar toda la información necesaria para ser entendidos.

Los servicios web comunican aplicaciones, lo cual permite que se comparta información independientemente de cómo se hayan creado, cuál sea el sistema operativo o la plataforma en que se ejecutan y cuáles sean los dispositivos utilizados para obtener acceso a ellas. La comunicación se caracteriza por el intercambio de mensajes XML y por ser independientes del protocolo de comunicación. Para lograr esta independencia, el mensaje XML se envuelve de manera apropiada para cada protocolo gracias a la creación del protocolo de transporte SOAP (*Simple Object Access Protocol: Protocolo de Acceso a Objetos Simples*). (Alvez, et al., 2006)

El Lenguaje de Descripción de los Servicios Web (WSDL: *Web Services Description Language*), expresado en XML, describe cómo acceder al servicio, de qué funciones dispone, qué argumentos necesita y qué devuelve cada uno.

La otra tecnología fundamental implicada es la de Descripción Universal, Descubrimiento e Integración (UDDI: *Universal Description, Discovery, and Integration*). UDDI es un directorio de servicios web donde se puedan publicar los servicios ofrecidos, dar características del tipo de servicio, y realizar búsquedas.

En resumen, SOAP define un protocolo XML para la interoperabilidad básica entre servicios, WSDL introduce un lenguaje común para describir servicios y UDDI provee la infraestructura requerida para publicar y descubrir servicios programáticamente. Juntas, estas especificaciones permiten a las aplicaciones interactuar siguiendo un modelo débilmente acoplado e independiente de la plataforma subyacente.

### 1.4.3.1. Tecnologías asociadas

#### WSDL

El lenguaje de descripción de servicios web, nació en septiembre de 2000 de la mano de *Microsoft*, *IBM* y *Ariba*, y constituye un estándar para la descripción de servicios del *World Wide Web Consortium* (W3C), organización que guía el desarrollo en la web. (Endrei, et al., 2004)

En esencia, WSDL es un contrato entre el proveedor del servicio y el cliente mediante el que el proveedor del servicio indica (Alvez, et al., 2006):

- ◆ Qué funciones que se pueden invocar
- ◆ Qué tipos de datos utilizan esas funciones
- ◆ Qué protocolo de transporte se utilizará para el envío y recepción de los mensajes (típicamente, pero no únicamente, mensajes SOAP).
- ◆ Cómo acceder a los servicios. En esencia, mediante qué URL se utilizan los servicios.

#### SOAP

Es un protocolo simple para el intercambio de información estructurada en un entorno distribuido y descentralizado. Define como dos objetos en diferentes procesos pueden comunicarse por medio del intercambio de datos XML. Utiliza XML para definir un framework extensible de mensajería proveyendo un formato de mensaje que puede ser intercambiado sobre una variedad de protocolos subyacentes. El framework fue diseñado para ser independiente de cualquier modelo de programación o cualquier semántica específica de alguna implementación (W3C, 2007).

El hecho de que SOAP use XML tiene sus ventajas como la facilidad de comprensión por las personas, pero a la vez tiene sus inconvenientes debido a que los mensajes son más largos.

## UDDI

Es un elemento central del grupo de estándares involucrados en la tecnología servicios web. Es el mediador a través del cual se conocen los posibles clientes con los proveedores de los servicios. Define un método estándar para publicar y descubrir servicios en el contexto SOA (Alvez, et al., 2006).

La especificación de UDDI nació casi a la vez que la de WSDL, de la mano de las mismas compañías. La versión actual es la 3.0, especificación que data de agosto de 2003, siendo gestionada por *Organization for the Advancement of Structured Information Standards* (OASIS). La implementación de estas especificaciones se denomina “Registro UDDI”, el cual proporciona un conjunto de servicios web de registro y consulta vía SOAP.

El propósito funcional de un registro UDDI es la representación de datos y metadatos acerca de servicios web. Tanto para ser usado en una red pública como dentro de la infraestructura interna de una organización, un registro UDDI ofrece un mecanismo basado en estándares para clasificar, catalogar y manejar servicios web de forma de que puedan ser descubiertos y consumidos por otras aplicaciones.

### Estructuras de datos en UDDI

La información almacenada en un registro UDDI es un conjunto de las denominadas “estructuras de datos UDDI”. Estas estructuras, en XML, definidas en la especificación, son las que el cliente intercambiará con el registro UDDI. Cada instancia de estas estructuras se identifica de manera única con un identificador denominado *Universal Unique Identifier* (UUID) (OASIS, 2004).

Las principales estructuras de alto nivel son las siguientes (Alvez, et al., 2006):

- ◆ BusinessEntity. Contiene información básica de la empresa: persona de contacto, una clasificación de la empresa conforme a alguna de las taxonomías definidas, así como una descripción en lenguaje natural de las actividades de la empresa.
- ◆ PublisherAssertion. Una empresa puede declarar su relación con otras empresas, por ejemplo, como socios o como clientes. Cada una de estas relaciones se modela como una PublisherAssertion.
- ◆ BusinessService. Este elemento muestra los servicios ofrecidos por una empresa. Estos servicios pueden ser servicios web o no. Una BusinessEntity se compone de uno o varios elementos businessService, y un elemento businessService puede ser usado por varios elementos BusinessService.

- ◆ BindingTemplate. Este elemento contiene referencias a descripciones técnicas (por ejemplo, URLs apuntando a manuales técnicos) y a las URL de acceso de los servicios web. Cada elemento BusinessService puede tener uno o varios elementos BindingTemplate.
- ◆ TModel. Este elemento describe la manera de interactuar con el servicio web y su comportamiento. Entre sus datos se encuentra la URL donde se encuentra el documento WSDL. También pueden aparecer aquí las categorías en las que se puede englobar el servicio (pueden ser distintas de las que podían aparecer en BusinessEntity).

#### 1.4.4. Composición de Servicios

En situaciones en las que un sistema de software, para completar un proceso de negocio, tiene que interactuar con otros sistemas independientemente de su distribución física y heterogeneidad, las Arquitecturas Orientadas a Servicios son más que útiles.

De las Arquitecturas Orientadas a Servicios y más precisamente del uso de servicios web surge un nuevo concepto denominado *composición de servicios*. Esta composición no solo permite modelar el proceso de negocio sino que también maximiza las potencialidades que SOA brinda a través de la integración de datos y aplicaciones.

La composición de servicios implica encontrar un mecanismo que permita a dos o más de ellos cooperar entre sí para resolver requerimientos que van más allá del alcance de sus capacidades individuales. Algunos de los requisitos básicos que se deben cumplir son (Alvez, et al., 2006):

- ◆ **Interacciones asincrónicas con servicios:** de manera de dar la posibilidad de crear procesos que transcurran durante largos períodos de tiempo.
- ◆ **Interacciones simultáneas entre servicios:** esto introduce el procesamiento en paralelo lo cual redundará en un aumento considerable de la eficiencia.
- ◆ **Manejo de excepciones:** un proceso basado en múltiples servicios puede fallar en su ejecución si al menos uno de sus componentes falla. Se debe tener en cuenta la ocurrencia de errores y proveer mecanismos para manejarlos.
- ◆ **Integridad transaccional:** un proceso de larga duración puede eventualmente fallar, en este caso puede requerirse que parte de las acciones ya efectuadas se deshagan.

Dos términos comúnmente usados para referirse a la colaboración entre servicios son orquestación y coreografía. Ambos están directamente relacionados con la composición pero se enfocan en aspectos complementarios de la interacción entre servicios.



### Orquestación

Un proceso se puede considerar una orquestación de servicios cuando es controlado totalmente por una única entidad. Este proceso define completamente las interacciones con los servicios componentes y la lógica requerida para conducir correctamente esas interacciones. Este tipo de proceso puede entenderse como privado y ejecutable ya que solo la entidad que está orquestando el proceso conoce el flujo de control e información que sigue el proceso que se está orquestando. De esta manera se crea un proceso que utiliza diferentes servicios manipulando la información que fluye entre ellos, convirtiendo, por ejemplo, los datos de salida de algunos servicios en datos de entrada de otro. Aquí, cada entidad que participa implementa y controla su propio proceso (Cubillos, et al., 2004).

### Coreografía

Un proceso es una coreografía de servicios cuando define las colaboraciones entre cualquier tipo de aplicaciones componentes, independientemente del lenguaje o plataforma en el que estén definidas las mismas. Un proceso de coreografía no es controlado por uno solo de los participantes. A diferencia de la orquestación, la coreografía puede verse como un proceso público y no ejecutable. Público porque define un comportamiento común que todas las entidades participantes deben conocer, y no ejecutable porque está pensado para verse más bien como un protocolo de negocio que dicta las reglas para que dichas entidades puedan interactuar entre sí (Cubillos, et al., 2004).

### 1.4.5. SOA & Web Service

Una SOA a menudo es malinterpretada y confundida con los servicios web. SOA es un acercamiento al diseño de sistemas, dirige como los recursos tecnológicos serán integrados y que servicios serán expuestos. A su vez los servicios web son una metodología de implementación que usa estándares específicos y protocolos de lenguaje para ejecutar la solución SOA.

Una Arquitectura Orientada a Servicios es un método de diseñar y construir soluciones de software poco acopladas que expongan sus funciones del negocio como servicios de software accesibles a través de programación para ser usadas por otras aplicaciones a través de la publicación de interfaces. Los servicios web representan una implementación de una Arquitectura Orientada a Servicios pero no todas las aplicaciones SOA pueden ser consideradas servicios web, tal es el caso de *Representational State Transfer* (REST).

A pesar de esto los servicios web son la tecnología actual, y al parecer, la tecnología que ha llegado para quedarse. De hecho WSDL sigue desarrollándose, y todas las tecnologías más allá del lenguaje de descripción que tiene que ver con servicios web siguen haciéndolo de manera muy vertiginosa.

Por otra parte las Arquitecturas Orientadas a Servicios son un concepto mucho más amplio que los servicios web, aunque estos sean la implementación actual.

Un servicio web es SOA si (Reynoso, 2005):

- ◆ Las interfaces se basan en protocolos de web (HTTP, SMTP, FTP)
- ◆ Los mensajes se basan en XML.

#### 1.4.6. Modelo de Madurez de SOA

##### ¿Qué es un modelo de Madurez?

- ◆ Permite medir el estado actual de una arquitectura empresarial respecto a la utilización de SOA.
- ◆ Permite establecer una ruta de evolución.

##### ¿Por qué un Modelo de Madurez?

- ◆ Habilita aprendizaje por capas incluyendo buenas prácticas
- ◆ Forma la base para comunicar y extender capacidades.
- ◆ Ayuda en la construcción de itinerarios.
- ◆ Forma la base para crear una adopción incremental de SOA.

El modelo de madurez propuesto en esta sección consta de cinco niveles: servicios iniciares, servicios arquitecturados, servicios de negocio, de colaboración, de métricas de negocio y optimizados de negocio. Estos niveles van a permitir ir escalando en ganancia de una SOA ya que en este tipo de arquitecturas se va de menos a más (Ver anexo 12). Este modelo de madurez es propuesto por *Progress Software Corporation* una reconocida empresa de software dedicada al suministro de plataformas de aplicaciones de negocios, entre ellas, infraestructuras SOA y de integración. Tiene productos en el mercado del software reconocidos como: SONIC ESB, entre otros.

## 1.5. ADLs y UML

### ADL

Los Lenguajes de Descripción de Arquitectura (ADL: *Architecture Description Language*) permiten modelar una arquitectura antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento. Los mismos son bien reconocidos por la comunidad académica de Arquitectura de Software, sin embargo parecen no gozar de la misma aceptación en la práctica industrial, al no utilizarlos en sus proyectos de software.

Estos suministran construcciones para especificar abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo familias de arquitecturas o estilos. (Reynoso, et al., 2004)

Los ADLs están caracterizados por la abstracción que proveen al usuario, además la mayoría de las vistas provistas por estos lenguajes contienen información predominantemente arquitectónica y el análisis provisto por el lenguaje se fundamenta en información de nivel arquitectónico.

**Algunas de las ventajas que proponen los ADLs son:**

- ◆ Es posible hacer la descripción del comportamiento y sus elementos asociados, tales como el tipo de eventos que producen, o a los que responden, incluyendo descripciones o documentación de alto nivel.
- ◆ Facilidad con la que puede introducirse y mantenerse la información referente al sistema.
- ◆ Los componentes pueden ser refinados en la medida que sea necesario, para distintos tipos de análisis.

A pesar de todo lo anterior expuesto se puede decir que los ADLs son convenientes, pero no han demostrado aún ser imprescindibles, sobre todo por la permanencia de UML, ahora con la versión 2.0 (Reynoso, et al., 2004).

**UML**

El Lenguaje de Modelado Unificado (UML: *Unified Modeling Language*) está especializado en construir, especificar, visualizar y documentar los elementos que se producen en el proceso de desarrollo de software de Sistemas de Software Orientados a Objetos. Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas y debido a que es un lenguaje, cuenta con reglas para combinar tales elementos. Ofrece soporte para clases, relaciones, comportamiento por interacción, empaquetamiento, entre otros. Estos elementos se pueden representar mediante varios diagramas ofrecidos por este lenguaje que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo.

UML a pesar de no calificar en absoluto como ADL, se ha probado que puede utilizarse no tanto como un ADL por derecho propio, sino como metalenguaje para simular otros ADLs, y en particular C2 y Wright (Reynoso, et al., 2004).

En UML se identifican:

- ◆ Elementos (abstracciones que constituyen los bloques básicos de construcción)
- ◆ Relaciones (Ligan los elementos)
- ◆ Diagramas (Representación gráfica de un conjunto de elementos)

UML provee una notación para la descripción de la proyección de los componentes de software en el hardware, correspondiendo con la vista física del modelo 4+1 de Kruchten para la descripción de arquitecturas de software (Kruchten, 1999).

En UML existe soporte para algunos de los conceptos asociados a las arquitecturas de software, como los componentes, los paquetes, las librerías y la colaboración por lo que se puede decir que los elementos básicos de la arquitectura se pueden modelar muy bien con UML. Pero para una representación total de la arquitectura harían falta otras herramientas y lenguajes, pues la representación total no es solo la comunicación que existe entre sus componentes, también se deben documentar y justificar todo lo realizado para lograr un buen diseño arquitectónico.

## 1.6. Calidad en la Arquitectura de Software

La Arquitectura de Software de los Sistemas de Software a ser construidos, se convierte en un factor de importancia para lograr que éste tenga un alto nivel de calidad. El poseer una buena Arquitectura de Software es de suma importancia, ya que ésta es el corazón de todo Sistemas de Software y determina cuáles serán los niveles de calidad asociados al sistema.

De lo anterior se puede deducir que los objetivos de evaluar una arquitectura es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders y en qué grado lo hace. Además de analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante.

Cabe señalar que los requerimientos no funcionales también son llamados atributos de calidad. Un atributo de calidad es una característica de calidad que afecta a un elemento. Donde el término *característica* se refiere a aspectos no funcionales y el término *elemento* a componente (Gómez, 2007).

### Técnicas de Evaluación

Existen un grupo de técnicas para evaluar que se clasifican en cualitativas y cuantitativas (Brey, et al., 2005) (ver anexo 13):

- ◆ Técnicas de cuestionamiento o cualitativas. Utilizan preguntas cualitativas para preguntarle a la arquitectura
  - Cuestionarios. Abiertas. Temprana
  - Checklists. Especifico del Dominio de la aplicación.
  - Escenarios. Especificas del Sistema. Arquitectura avanzada.
- ◆ Measuring techniques. Sugiere hacerle medidas cuantitativas a la arquitectura.
  - Utiliza métricas arquitectónicas, como acoplamiento, cohesividad en los módulos, profundidad en herencias, modificabilidad.
  - Simulaciones, Prototipos, y Experimentos

Por lo regular, las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada (Gómez, 2007).

#### ¿Por qué cualidades puede ser evaluada una Arquitectura?

A grandes rasgos, Bass establece una clasificación de los atributos de calidad en dos categorías (Camacho, et al., 2004):

- ◆ **Observables vía ejecución:** aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en el (anexo 14).
- ◆ **No observables vía ejecución:** aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en el (anexo 15).

#### ¿Qué resultados produce la evaluación de una Arquitectura?

Una vez que se ha efectuada la evaluación se debe elaborar un reporte. Que debe presentarse como un documento preliminar, con la finalidad de que se corrija por las personas que participaron en la evaluación. El contenido del reporte responde a dos tipos de preguntas (Gómez, 2007):

- ◆ ¿Se ha diseñado la arquitectura más apropiada para el sistema?
- ◆ ¿Cuál de las arquitecturas propuestas es la más apropiada para el sistema a construir?

Además de responder estas preguntas, el reporte también indica:

- ◆ El grado en que se cumplieron los atributos de calidad.
- ◆ Lista priorizada de los atributos de calidad requeridos para la arquitectura que está siendo evaluada.
- ◆ Riesgos y no riesgos.

Hoy en día existen varios métodos para realizar evaluaciones a una Arquitectura de Software que verifican el cumplimiento o no de ciertos atributos de calidad, siendo unos más específicos que otros, como es el caso de: *Architecture Trade-off Analysis Method* (ATAM), Bosch (2000), *Active Design Review* (ADR), *Active Reviews for Intermediate Design* (ARID), Losavio (2003).

Existen otros métodos de evaluación de arquitectura que evalúan un atributo de calidad específico, como: *Architecture Level Modifiability Analysis* (ALMA), *Performance Assessment of Software Architecture* (PASA), *Scenario based Architecture Level Usability Analysis* (SALUTA) y *Survivable Network Analysis* (SNA).

Independientemente de cuan específico o no pueda ser el método de evaluación la gran mayoría tiene algo en común y es que utilizan la técnica de escenarios como vía de constatar en qué medida la arquitectura responde a los atributos de calidad requeridos por el sistema.

Un método de evaluación no es mejor que otro, sino que evalúa mejor, en ciertas condiciones, un atributo de calidad dado. Por lo que se concluye que en dependencia de las condiciones y lo que se desea evaluar, será el método de evaluación empleado (que no tiene que ser solo uno).

### **Relación entre la Arquitectura de Software y Atributos de Calidad**

Durante el proceso de desarrollo de todo sistema de software se toman decisiones de diseño arquitectónico que permiten alcanzar ciertos y determinados atributos de calidad. Para esto el arquitecto se debe cuestionar frecuentemente cuál será el impacto que esa decisión causara sobre otros atributos de calidad.

Cada decisión incorporada en una arquitectura de software puede afectar potencialmente los atributos de calidad (Bass, et al., 2000). Por lo que se puede decir que al tomar dichas decisiones generalmente se tienen consecuencias como:

- ◆ Formulación de argumentos que expliquen cómo la decisión tomada permite alcanzar los atributos de calidad propuestos.
- ◆ Preguntas referentes al impacto de tomar dicha decisión, sobre otros atributos de calidad.

La relación que existe entre los Atributos de Calidad y la Arquitectura de Software tiene diversos beneficios (Camacho, et al., 2004):

- ◆ Realza en gran medida el proceso de análisis y diseño arquitectónico, ya que el arquitecto puede reutilizar análisis existentes y determinar acuerdos explícitamente en lugar de hacerlo sobre la marcha.

- ◆ Una vez que el arquitecto entiende el impacto de los componentes arquitectónicos sobre uno o varios atributos de calidad, estaría en capacidad de reemplazar un conjunto de componentes por otro cuando lo considere necesario.
- ◆ Una vez que se codifica la relación entre arquitectura y atributos de calidad, es posible construir un protocolo de pruebas que habilitará la certificación por parte de terceros.

### 1.7. Conclusiones parciales

A partir de los objetivos propuestos para este capítulo, se puede concluir lo siguiente:

A lo largo de este capítulo se ha expuesto los principales aspectos dentro de la disciplina como: concepción de las principales definiciones, tendencias actuales, estilos más novedosos y reinantes en el mundo de las aplicaciones empresariales, exponiendo sus características, ventajas y desventajas. Se analizaron los patrones arquitectónicos y niveles de abstracción de la arquitectura que ayudaron a comprender conceptos como: estilos arquitectónicos y marcos de trabajo. También se ha abordaron temas como la calidad en la Arquitectura de Software definiéndose los principales atributos de calidad y métodos empleados para constatar en que medida la arquitectura cumple con los parámetros de calidad que se establezcan. De igual forma se analizaron las Arquitecturas Orientadas a Servicios como máximo exponente del presente que permitió adoptarlo para aplicarlo en la propuesta de solución ya que brindará una mayor flexibilidad al sistema ante el negocio cambiante que experimentan las empresas hoy en día y a la facilidad de integración que brinda este tipo de arquitectura.

## CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

### 2.1. Introducción

En este capítulo se realiza la propuesta de solución al problema planteado en el diseño teórico de la investigación, la misma viene dada por el artefacto Documento Descripción de Arquitectura que propone la Universidad de las Ciencias Informáticas a emplear en los proyectos productivos. Siendo dicho artefacto de gran importancia ya que proporciona, entre otras cosas, una vista arquitectónica del sistema a todos los implicados en el mismo.

### 2.2. Descripción de la Arquitectura

#### 2.2.1. Introducción

##### 2.2.1.1. Propósito

Este documento identifica la arquitectura propuesta para el Sistema de Gestión de Inventario a partir las 4+1 vistas arquitectónicas propuestas por la metodología RUP, las mismas facilitan la comprensión de la arquitectura propuesta y aportan argumentos para la identificación de los elementos claves del sistema en desarrollo, proporcionando información al cliente sobre las decisiones arquitectónicas en la construcción del producto.

Además este documento persigue los siguientes propósitos fundamentales:

- ◆ Proporcionar una comprensión arquitectónica global del sistema a los stakeholders, mediante el uso de de vistas arquitectónicas, de modo que se muestre los aspectos más significativos del sistema.
- ◆ Proporcionar soporte para toma de decisiones arquitectónicamente significativas que deban ser tomadas en el desarrollo del sistema.
- ◆ Organizar el desarrollo del sistema.
- ◆ Fomentar la reutilización.
- ◆ Hacer evolucionar el sistema.

##### 2.2.1.2. Alcance

El alcance del presente documento se extiende a todo el Sistema de Gestión de Inventarios, a todos los integrantes del proyecto encargados de desarrollarlo ya que el mismo influye en la toma de decisiones arquitectónicas, y a todos los implicados en el desarrollo del sistema como: clientes, para



que comprendan con suficiente detalle qué se está haciendo y cómo, de modo que facilite su participación.

### 2.2.2. Representación Arquitectónica

A partir del estudio de los estilos arquitectónicos realizado en el capítulo 1 y de las características particulares del sistema a desarrollar, siendo el mismo un módulo del ERP cubano; como parte de las decisiones arquitectónicas para definir la propuesta de arquitectura se decidió utilizar para el Sistema de Gestión de Inventarios una Arquitectura Orientada a Servicios. El por qué de esta decisión viene dado por diferentes razones:

- ◆ Porque las Arquitecturas Orientadas a Servicios permiten responder más rápidamente a las condiciones cambiantes del negocio empresarial, promoviendo y permitiendo la reutilización, la interconexión de tecnologías existentes en vez de consumir tiempo y costos en la reinversión.
- ◆ Un Sistema de Gestión de Inventarios está formado por un conjunto de procesos que se ejecutan y que son reutilizables para otros módulos como el de contabilidad, comerciales, logística, entre otros.
- ◆ Por otra parte el Sistema de Gestión de Inventarios, al formar parte del ERP, necesitará intercambiar gran cantidad de información e interactuar con otros módulos del mismo.
- ◆ Es un estilo arquitectónico que se basa fundamentalmente en estándares de desarrollo de los servicios web como: XML, WSDL y SOAP, garantizando la interoperabilidad entre distintos servicios y aplicaciones que colaboran con la agilidad del negocio. Tiene la ventaja de ser un modelo arquitectural escasamente acoplado, lo cual le dota de una enorme flexibilidad para la definición, implementación, sustitución, evolución de los servicios y las funcionalidades que contiene.

Las Arquitecturas Orientadas a Servicios básicamente siguen la filosofía de un proveedor-consumidor, auxiliándose de un repositorio *UDDI* que brinda servicios de directorio, por llamarlo de alguna manera, ya que los proveedores publican las descripciones de los servicios en el repositorio de servicios, que para la solución se propone usar *JUDDI* que es una implementación libre de *UDDI* para java, y cuando el consumidor necesite de algún servicio consultará ese repositorio para encontrar la información necesaria para localizar dicho servicio, si existe, para su posterior invocación. La propuesta de solución no estará exenta de ese principio tan básico de este tipo de arquitecturas, que muestra como colaboran las aplicaciones en una Arquitecturas Orientadas a Servicios.



Figura 1. Colaboración en una Arquitecturas Orientadas a Servicios

La propuesta de solución, según el modelo de madurez analizado en el capítulo anterior, estará enmarcada en un primer nivel con elementos del segundo, como son: la UDDI y WS-Security, garantizándose de esta forma la integración del sistema con cualquier otra aplicación o servicio que necesite información de los inventarios, así como la seguridad de los servicios web.

El sistema propuesto estará estructurado en subsistemas y módulos definidos de forma vertical que colaboran entre sí mediante interfaces y una estructuración por niveles lógicos (capas) de forma horizontal posibilitando así el intercambio de datos, al servirse el nivel superior del inferior y este a su vez brinda sus servicios a su inmediato superior, donde la capa de servicios compartidos expondrá los procesos de negocio mediante servicios web.

La estructuración del sistema en capas viene dada por las ventajas que propicia como son: la flexibilidad, escalabilidad, reutilización, capacidad de mantenimiento, entre otras, que coinciden, en gran parte, con las variables que se proponen los autores resolver con dicha solución.

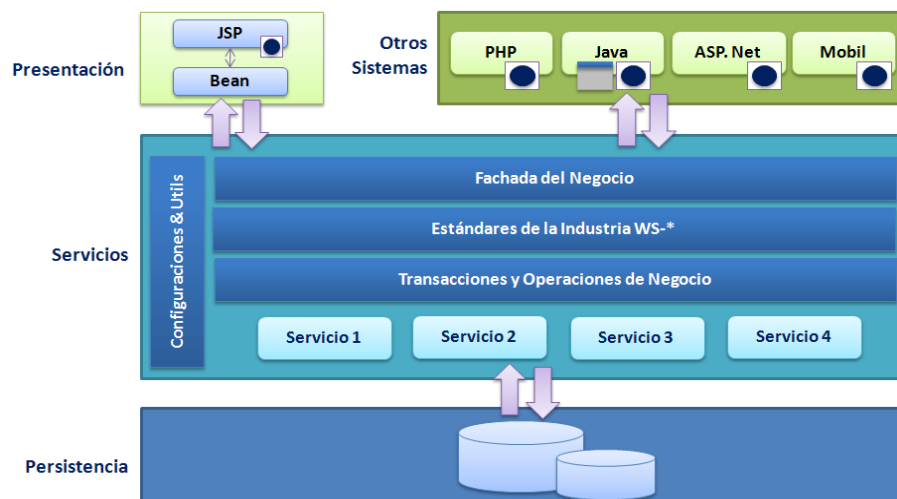


Figura 3. Arquitectura en Capas de la Propuesta SOA

### ❖ Capa de Presentación

Esta capa es la encargada de gestionar los datos de entrada y salida mediante las interfaces de usuario, que van a permitir la interacción del sistema con los usuarios potenciales. En otras palabras, maneja el contexto del usuario e interactúa con la capa de negocio donde está implementada toda la lógica de la aplicación.

A partir de los Requisitos no Funcionales de los clientes y de las facilidades que brinda el diseño arquitectónico propuesto, la capa de presentación del Sistema de Gestión de Inventarios puede ser desarrollada en cualquier ambiente, tanto *Web como Desktop*. Finalmente se decidió desarrollar la capa de presentación sobre la Web haciendo uso del framework *MyFaces*. De esta forma se busca abaratar la solución mediante el uso de clientes ligeros, los cuales no ejecutan demasiadas labores de procesamiento para la ejecución de la aplicación misma.

Es válido destacar que la decisión de realizar la aplicación web también viene dada por el concepto de no dejar que el cliente realice demasiadas tareas, solo lo necesario para que lleve a cabo su trabajo y dejar que en el lado del servidor se realicen las operaciones importantes: almacenamiento de datos, transacciones, reglas del negocio y la lógica del programa.

Entre las ventajas de las aplicaciones basadas en la web se pueden mencionar:

- ◆ Compatibilidad multiplataforma
- ◆ Actualizaciones al sistema son más rápidas y fáciles, pues basta con realizar los cambios sobre el servidor donde este corriendo la aplicación y todos los clientes dispondrán de ella una vez que accedan al sistema.
- ◆ Acceso inmediato vía cuenta online.
- ◆ Facilidad de prueba.
- ◆ Menores requerimientos de memoria local.
- ◆ Precio ya que consumen menor cantidad de recursos.
- ◆ Múltiples usuarios concurrentes.

Por otra parte, si bien es cierto que la arquitectura cliente servidor de la web ha ofrecido muchas ventajas, también es cierto que carece de la riqueza gráfica de las aplicaciones de escritorio que cuentan con controles inteligentes que dan mayor fluidez al trabajo del usuario, esto ha sido resuelto con varias estrategias o tecnologías tales como *AJAX*, *FLASH*, *Web 2*, entre otras. Así que en vez de ir perdiendo fuerza debido a la pobreza en sus interfaces gráficas, la web busca alternativas que le

permitan ofrecer todas sus ventajas pero con la posibilidad de ofrecer controles visuales más amigables al trabajo del usuario.

#### ◆ Patrones

##### ○ Presentación

- **Nombre:** Modelo – Vista – Controlador (MVC)
- **Problema:** La lógica de un interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica de negocio. Si se realiza un diseño ofuscado, es decir, un pastiche que mezcle los componentes de interfaz y de negocio, entonces la consecuencia será que, cuando se necesite cambiar el interfaz, se tendrá que modificar trabajosamente los componentes de negocio. Mayor trabajo y más riesgo de error. (Lago, 2007).
- **Solución:** Se trata de realizar un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.
- **Elementos del patrón:**
  - ✓ Modelo: datos y reglas de negocio.
  - ✓ Vista: muestra la información del modelo al usuario.
  - ✓ Controlador: gestiona las entradas del usuario.

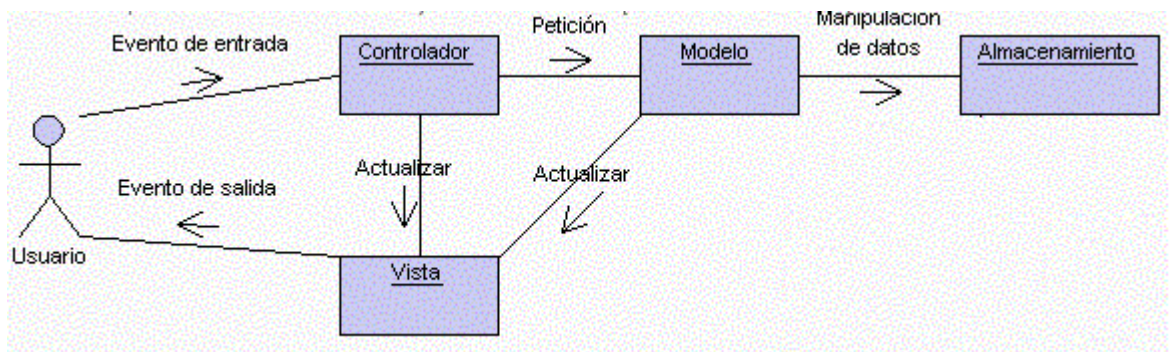


Figura 4. Estructura del patrón Modelo – Vista – Controlador (Lago, 2007)

#### ❖ Capa de Servicios

Una característica peculiar de las arquitecturas en capas es que las capas inferiores brinden las funcionalidades que necesitan las capas inmediatamente superiores, y a la vez estas, solo puedan acceder a las funcionalidades proporcionadas por las capas inmediatamente inferiores. Visto de esta manera, la capa de servicios es la encargada proporcionar las funcionalidades que necesita la capa de

presentación. Además es la responsable de gestionar toda la lógica de negocio de la aplicación y a la vez representa el contenedor lógico donde se ubican los servicios compartidos.

Las transacciones y operaciones de negocio serán realizadas por los servicios web que son los encargados de exponer toda la funcionalidad del sistema. De tal forma que los procesos de negocio del sistema de gestión de inventario serán expuestos mediante servicios web compartidos que podrán ser accedidos por otros servicios o sistemas que necesiten información de los inventarios.

Para el desarrollo de cada servicio web se utilizarán los estándares de la industria, entre ellos los: *XML* como formato estándar para los datos que se vayan a intercambiar, *SOAP* como protocolo para el intercambio de datos, *WSDL* para la descripción de la interfaz pública de los servicios web, *WS-Security* para la autenticación de los actores y la confidencialidad de los mensajes enviados y finalmente la *UDDI* para publicar la información de los servicios web y comprobar qué servicios web están disponibles.

Otro aspecto común a todos los servicios que se desarrollen, será crearle una fachada al negocio. Crear esta fachada tiene como objetivo diferenciar los objetos del negocio de los objetos que necesitan los servicios web, ya que es una mala práctica usar los mismos objetos para las dos cosas, debido a que, quien quiera que utilice el servicio web no tiene por qué conocer más allá de la información necesaria para utilizarlo.

A lo largo de esta capa habrá un conjunto de configuraciones y utilidades comunes para todos los servicios que se desarrollen, que se encargan, entre otras cosas, de la interconexión entre las diferentes capas vía XML.

Las Arquitecturas Orientadas a Servicios se distinguen por exponer la lógica del negocio mediante servicios web que deben poder ser accedidos por todas aquellas aplicaciones que necesiten de ellos. La vía más utilizada hoy en día para acceder a estos servicios es a través de un registro UDDI, mediante el cual las aplicaciones pueden publicar y descubrir dichos servicios para su invocación.

### **JUDDI**

Es una implementación libre de la especificación UDDI para java en cuestiones de servicios web. Entre sus principales características están: que es open source, independiente de plataforma, brinda soporte para JDK 1.3.1 hasta JDK 1.5, puede usarse con cualquier base de datos relacional que soporte *Structured Query Language (SQL)* estándar como: *MySQL*, *DB2*, *Sybase*, *JDataStore*, *HSQLDB*. Es desplegable en cualquier servidor de aplicación de java que soporte la especificación *Servlet 2.3* como:

*Jakarta Tomcat, JOnAS, WebSphere, WebLogic, Borland Enterprise Server, JRun*. Es de fácil integración con sistemas de autenticación existentes. Soporta configuración desplegable de clúster de registros de JUDDI. (Apache, 2007).

La gran mayoría de las empresas que utilizan UDDI lo hacen en conjunto con un Bus de Servicios Empresarial (*ESB: Enterprise Service Bus*), ya que este facilita la integración con otras aplicaciones, sirve de mediador para la orquestación de servicios, maneja seguridad, en fin constituye un componente que ofrece gran valor.

Hoy en día muchas empresas desarrollan su propio ESB para adaptarlo a las prestaciones de sus modelos, pero este proceso implica mucho tiempo y esfuerzo. Por otra parte utilizar un ESB genérico es complicado puesto a que la curva de aprendizaje es lenta, se necesita escribir mucho código y desarrollar componentes para enlazarlos con otros que no son los desarrollados por el fabricante original.

El Sistema de Gestión de Inventario en esta primera versión, manejará la lógica transaccional internamente dentro de los servicios web a nivel de servicio de negocio con el framework Spring, por lo que no se necesitará de un mediador (ESB), ni de un Lenguaje de Ejecución de Procesos de Negocio (*BPEL: Business Process Execution Language*). Esto es algo costoso en el tiempo, pero en la medida en que se vaya ganando en madurez de SOA se podrá incluir estos componentes que aportan gran funcionalidad, flexibilidad y potencialidad al sistema.

De este modo se aprovechará las ventajas de integración que ofrece la UDDI ya que cualquier sistema de cualquier entidad que necesite comunicarse o interactuar con el módulo de inventario podrá hacerlo a través del repositorio de servicios, el cual brinda información sobre las funcionalidades ofrecidas y cómo acceder a ellas.

Algunos de los beneficios que se derivan del uso de UDDI son (Mateu, 2003):

- ◆ Posibilita compartir, buscar y reutilizar servicios web, así como otros recursos programables.
- ◆ Define cómo interactuar con el servicio escogido, una vez localizado éste.
- ◆ Permite llegar a nuevos clientes y facilita y simplifica el acceso a los clientes ya existentes.
- ◆ Describe los servicios y componentes o métodos de negocio de forma automática (o automatizable) en un entorno seguro, abierto y simple.
- ◆ Contribuye al aumento de la confiabilidad de las aplicaciones, así como la mejora del proceso de administración de estas.

- ◆ Contribuye al aumento de la productividad en menor tiempo, ya que permite reutilizar recursos ya elaborados.

### Vista interna de un servicio web

Los servicios web proporcionados por la capa de servicios, que expondrán toda la funcionalidad del sistema, serán también desarrollados en capas. A partir esta organización estructural en capas se propone la utilización de un framework por cada una de ellas, pues contienen funcionalidades bien definidas para un determinado dominio de la aplicación, un conjunto de componentes implementados e interfaces, que se pueden utilizar, redefinir y crear nuevos. De modo que se puedan crear los componentes necesarios por cada capa del servicio web.

- ◆ **Motor de Servicios Web:** Apache Axis2 1.3
- ◆ **Capa de Negocio:** Spring 2.0.
- ◆ **Capa de Acceso a Datos:** Hibernate 3.2.0

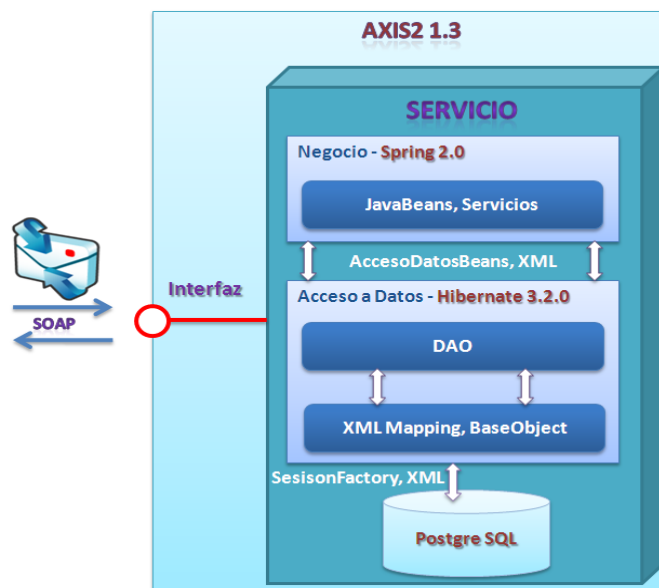


Figura 5. Vista interna de un Servicio Web

Los niveles lógicos definidos de forma horizontal abstraen al nivel superior de un grupo de funcionalidades. El nivel más bajo se encarga de la lógica de acceso a datos el cuál reutiliza las funcionalidades del framework Hibernate. En este nivel se encuentran: los Objetos de Negocio, los Ficheros de Mapeo y los Objetos de Acceso a Datos que se comunican con la base de datos mediante las Fábricas de Sesiones vía XML.

El nivel intermedio se auxilia del framework Spring. Contiene los JavaBeans y los servicios encargados de realizar las operaciones de negocio, para esto se auxilian de los Objetos de Acceso a Datos donde

mediante la inyección de dependencia se le inyecta la implementación del DAO correspondiente vía XML. Paralelamente a esto se emplea Acegi, que es un módulo de Spring, para gestionar la seguridad de forma no intrusiva mediante la programación orientada a aspectos.

Finalmente Axis2 se integra con Spring como motor de servicios web y herramienta para la comunicación de los servicios web remotos.

Los frameworks utilizados brindan la ventaja de integrarse perfectamente unos con otros mediante XML, posibilitando así, que fluya la comunicación entre ellos y la integración entre los componentes de las diferentes capas.

#### ◆ Patrones

##### ○ Negocio

- **Nombre:** Service Locator (Localizador de Servicio) (Microsystem, 2006)
- **Problema:** En más de una ocasión un cliente deberá hacer uso de *JNDI* ya sea para obtener una conexión a la base de datos, una referencia a la clase home de un *Enterprise Bean*, o una referencia a los canales de mensajería. Al ser algo común para muchas componentes, tiende a aparecer código similar múltiples veces y la creación repetida del objeto *InitialContext* que puede tomar cierto tiempo.
- **Solución:** Consiste en utilizar un objeto Service Locator para abstraer toda la utilización *JNDI* y para ocultar las complejidades de la creación del contexto inicial, de búsqueda y recreación de objetos home *EJB*. Varios clientes pueden reutilizar el objeto Service Locator para reducir la complejidad del código, proporcionando un único punto de control. (Alur, et al., 2001)
- **Elementos del patrón:**
  - ✓ Client: Este es el cliente del Service Locator y representa a cualquier parte de la aplicación que necesite acceder a un servicio determinado.
  - ✓ ServiceLocator: El *ServiceLocator* abstrae el API de los servicios de búsqueda (nombrado), las dependencias del vendedor, las complejidades de la búsqueda, y la creación de objetos de negocio, y proporciona una interface simple para los clientes.
  - ✓ InitialContext: El objeto *InitialContext* es el punto de inicio para los procesos de búsqueda y creación.
  - ✓ ServiceFactory: El objeto *ServiceFactory* representa un objeto que proporciona control del ciclo de vida para objetos *BusinessService*.



- ✓ BusinessService: *BusinessService* es un rol que cumple el servicio que el cliente ha solicitado. El objeto *BusinessService* se crea, se busca o se elimina mediante el objeto *ServiceFactory*.

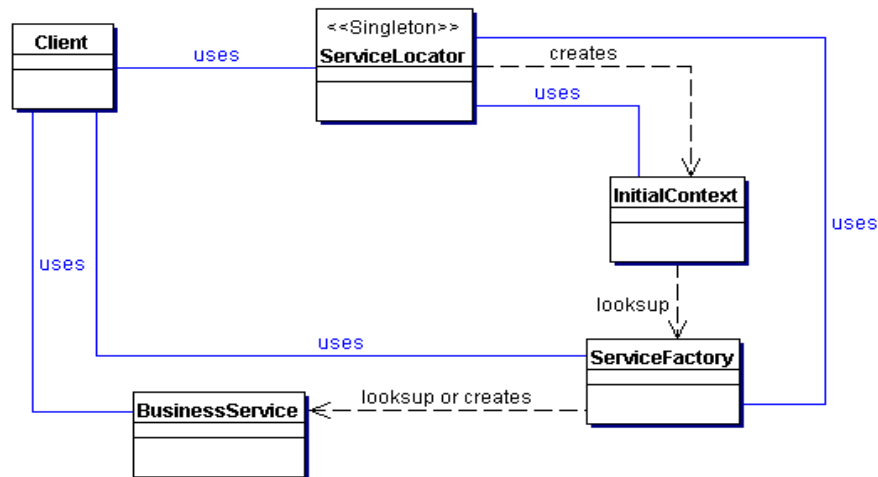


Figura 6. Estructura del patrón Service Locator (Microsystem, 2006)

#### ◆ Acceso a Datos

- **Nombre:** Data Access Object (Objeto de Acceso a Datos) (DAO) (Microsystem, 2006).
  - **Problema:** Este patrón surge de la necesidad de gestionar una diversidad de fuentes de datos, aunque su uso se extiende al problema de encapsular no sólo la fuente de datos, sino además ocultar la forma de acceder a los datos. Se trata de que el software cliente se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cual es la fuente de almacenamiento.
  - **Solución:** Utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos. Los componentes de negocio que tratan con el DAO utilizan un interface simple expuesta por el DAO para sus clientes. Como el interface expuesto por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio.
  - **Elementos del patrón:**
    - ✓ BusinessObject: Representa los datos del cliente. Es el objeto que requiere el acceso a la fuente de datos para obtener y almacenar datos.

- ✓ DataAccessObject: Es el objeto principal de este patrón. El mismo abstrae la implementación del acceso a datos subyacente al BusinessObject para permitirle un acceso transparente a la fuente de datos.
- ✓ DataSource: Representa la implementación de la fuente de datos.
- ✓ TransferObject: Representa un *TransferObject* utilizado para el transporte de datos. Además puede ser usado por un *DataAccessObject* para devolver y recibir los datos del cliente.

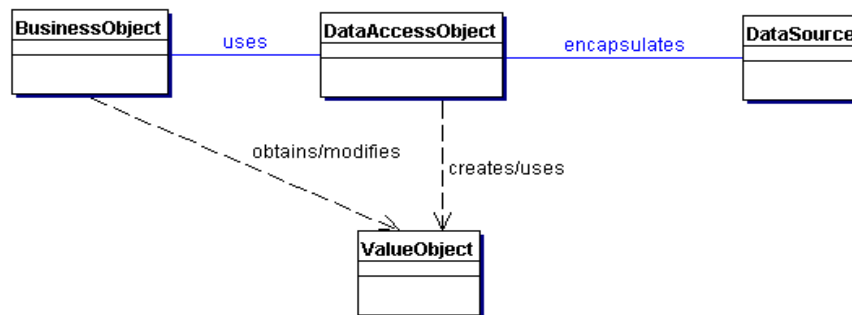


Figura 7. Estructura del patrón Data Access Object (DAO) (Microsystem, 2006)

#### ❖ Capa de Persistencia

La capa de persistencia es la encargada de gestionar el almacenamiento de los datos en el sistema gestor de base de datos. Siendo los componentes de acceso a datos una parte fundamental en el desarrollo del sistema.

La persistencia será manejada mediante software ORM (*Object Relational Mapping*) como frameworks de persistencia. Los ORM proporcionan:

- ◆ Mapear clases a tablas: propiedad columna, clase a tabla.
- ◆ Persistir objetos.
- ◆ Recuperar objetos persistidos.
- ◆ Recuperar una lista de objetos.

#### Sistema Gestor de Base de Datos (SGBD)

El Módulo de Inventarios, como todo Software de Gestión que es, necesita un repositorio de información, donde almacenar los datos necesarios para realizar las distintas operaciones.

La estructuración del sistema en Capas permite que se abstraiga la lógica de negocio del almacenamiento de los datos, la capa de acceso a datos contiene toda la lógica de acceso, ya sea consultas y procedimientos almacenados, dejando a la Base de Datos como simple almacén de datos sin ninguna lógica.

El cambio en el SGBD no costaría un esfuerzo en el desarrollo del sistema. Además se implementaría los mismos disparadores (*Triggers*) para mantener la seguridad y la confiabilidad en los datos, y los roles de usuario para cada una de las opciones.

Finalmente para terminar con la representación de la arquitectura el artefacto Documento Descripción de la Arquitectura se apoyará también en las 4+1 vistas de Philippe Krutchen, que responden a la tendencia: “*Arquitectura como etapa de ingeniería y diseño orientada a objetos*”. Siendo estas: la Vista de Caso de Uso, Vista Lógica, Vista de Procesos, Vista de Implementación, Vista de Despliegue.

Para la modelación de estas vistas se utilizó *UML* y *Visual Paradigm* como herramienta de modelado.

### 2.2.3. Objetivos y Restricciones Arquitectónicas

Las metas y las restricciones del sistema en cuanto a seguridad, portabilidad, escalabilidad e integrabilidad significativas para la arquitectura son:

#### ❖ **Requerimientos de Hardware**

##### ◆ **Estaciones de Trabajo**

- Tener periféricos Mouse y Teclado.
- Tarjeta de red.
- 256 Mb de memoria RAM.
- *Uninterruptible Power Supply (UPS)*.

##### ◆ **Servidor UDDI**

- Tener periféricos Mouse y Teclado.
- Microprocesador con 1Mb de cache L2, 1 GB de memoria RAM.
- 2 GB de espacio libre en disco.
- Tarjeta de red.
- *Uninterruptible Power Supply (UPS)*.

##### ◆ **Servidor de Aplicaciones Apache**

- Tener periféricos Mouse y Teclado.
- Microprocesador con 1Mb de cache L2, 1 GB de memoria RAM.
- 2 GB de espacio libre en disco.
- Tarjeta de red.
- *Uninterruptible Power Supply (UPS)*.

##### ◆ **Servidor de SW**

- Tener periféricos Mouse y Teclado.

- Microprocesador con 1Mb de cache L2, 2 GB de memoria RAM.
- 2 GB de espacio libre en disco.
- Tarjeta de red.
- Uninterruptible Power Supply (*UPS*).

◆ **Servidor de Base de Datos**

- Tener periféricos Mouse y Teclado.
- Microprocesador con 1Mb de cache L2, 2 GB de memoria RAM.
- Disco Duro con capacidad de 160 GB o más.
- Tarjeta de red.
- Uninterruptible Power Supply (*UPS*).

❖ **Requerimientos de Software**

◆ **Estaciones de Trabajo**

- Sistema Operativo: Windows 98 o superior, Debian, Ubuntu.
- Navegador Web: Internet Explorer 5.0 o superior, Mozilla Firefox 2.0 o superior.

◆ **Servidor UDDI**

- Sistema Operativo: Windows 98 o superior, Debian, Ubuntu.
- Gestor de Base de Datos: PostgreSQL 8.2
- Servidor Web: Apache Tomcat 6.0.13

◆ **Servidor Apache**

- Sistema Operativo: Windows 98 o superior, Debian, Ubuntu.
- Servidor de Aplicaciones: Apache 2.2.4

◆ **Servidor de SW**

- Sistema Operativo: Windows 98 o superior, Debian, Ubuntu.
- Máquina Virtual de Java: Java Development Kit (JDK) versión 1.6
- Servidor Web: Apache Tomcat 6.0.13

◆ **Servidor de Base de Datos**

- Sistema Operativo: Windows XP o superior, Ubuntu, preferentemente Debian.
- Gestor de Base de Datos: PostgreSQL 8.2

❖ **Redes**

- ◆ La red existente en las instalaciones debe de soportar la transacción de paquetes de información de al menos unas 10 máquinas a la vez.

- ◆ Para hacer más fiable la aplicación debe de estar protegida contra fallos de corriente y de conectividad, para lo que se deberá parametrizar los tiempos para realizar copias de seguridad. Implementar las transacciones de paquetes en la red con el protocolo TCP/IP que permite la recuperación de los datos.

#### ❖ Seguridad

- ◆ Se hará un adecuado uso desde la seguridad brindada por el repositorio de servicios Web (*JUDDI*), donde estarán publicados y serán descubiertos los mismos, mediante el mecanismo de autenticación y otorgamiento de tokens implementados en *JUDDI* para el acceso de los usuarios.
- ◆ La comunicación de servicios web mantendrá su integridad y confidencialidad con el estándar *ws-security* a través del modulo *Rampart* del framework *Axis2*.
- ◆ Los servicios ofrecidos por el sistema no podrán ser ejecutados por sistemas no autorizados, de manera que exista un sistema de autenticación para los servicios web que sea transparente al usuario de las aplicaciones.
- ◆ Tanto la comunicación entre los componentes de la plataforma como el manejo de datos del sistema deberán encontrarse cifrados mediante algoritmos de encriptación, específicamente el manejo de la seguridad de claves de acceso de usuarios.
- ◆ La seguridad también será manejada desde la propia arquitectura de despliegue que se propone, ya que para cualquier servicio o aplicación externa acceder al repositorio de servicios y a los servidores web se deberá atravesar un firewall de software configurado de tal manera que por defecto deniegue todas las entradas, aceptando sólo las direcciones que se especifiquen explícitamente y el puerto por el cual se podrán conectar. A la base de datos sólo podrán acceder los servidores web, de modo que en el fichero de configuración del SGBD PostgreSQL se le definirá que servidores específicos interactuarán con él, que en este caso serán los servidores web. Por otra parte, al Sistema de Gestión de Inventario, independientemente de la autenticación, sólo podrán acceder aquellas máquinas clientes que se especifiquen explícitamente, pues para acceder al mismo habrá que atravesar un firewall de software configurado de la misma forma que anteriormente mencionado, denegando todo excepto lo que se le especifique explícitamente.

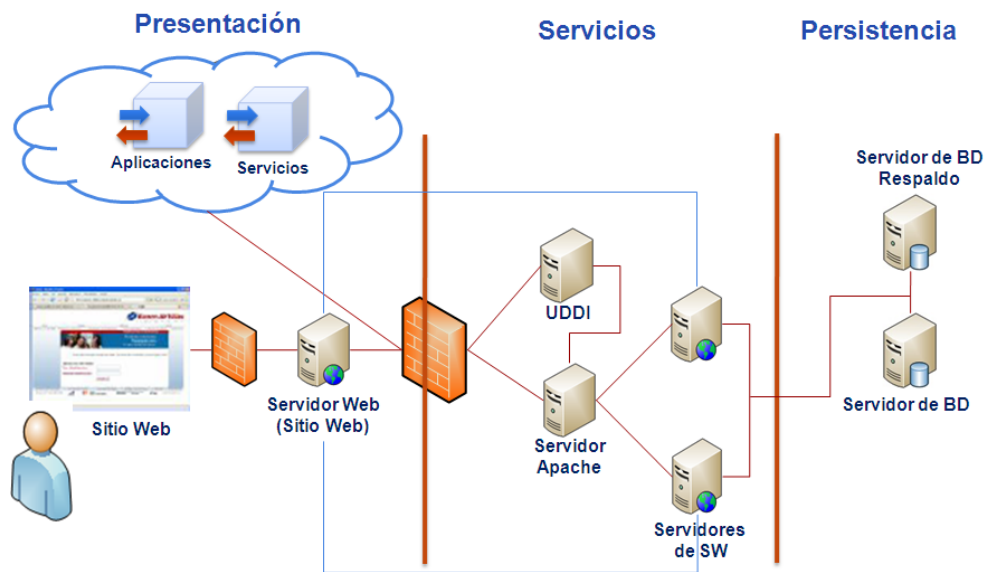


Figura 8. Propuesta de Despliegue

- ◆ Parte de la seguridad corre por parte de la tecnología Java utilizada para el desarrollo de la aplicación con el framework de seguridad Acegi que es un módulo de Spring.
- ◆ Se deberá utilizar usuarios de base de datos con roles bien definidos para cada una de las operaciones del sistema.
- ◆ Debe quedar constancia de quién, desde dónde, y cuándo se realizó una operación determinada en el sistema.
- ◆ Para garantizar la seguridad de la Base de Datos, no se permitirán privilegios de administración a ningún usuario.
- ◆ La asignación de usuarios y sus opciones sobre el sistema se garantizará desde el modulo de Administración del sistema.
- ◆ Programación de disparadores (*Triggers*) en la Base de Datos para no permitir la manipulación de los datos directamente en el SGBD.
- ◆ El sistema deberá generar copias de seguridad, periódicas y automáticas, de los datos contenidos en el mismo.
- ◆ Bajo petición de un usuario experto el sistema deberá permitir realizar copias de seguridad del sistema en el momento indicado.
- ◆ Se deberá permitir restaurar por parte de técnicos especializados la información del sistema a partir de una copia de seguridad anterior.

- ◆ El sistema gestionará los datos concurrentemente, de manera que al actualizar un registro, se comprobará si los datos que se quieren modificar no han sido modificados previamente por otro sistema.

#### ❖ **Portabilidad, escalabilidad, integrabilidad**

- ◆ El sistema deberá poder ser utilizado desde cualquier plataforma de software (Sistema Operativo).
- ◆ El sistema deberá hacer un uso racional de los recursos de hardware de la máquina, sobre todo en las PC clientes.
- ◆ La documentación de la arquitectura deberá ser reutilizable para poder documentarla como una familia de productos.
- ◆ Se desarrollará cada pieza del sistema en forma de componentes (elementos) con el fin de reutilizarlos para futuras versiones del sistema.
- ◆ El sistema deberá exponer toda su funcionalidad mediante servicios web que se registrarán en el directorio UDDI, garantizando así la integración de cualquier otro sistema o servicio que necesite información del Sistema de Gestión de Inventarios.
- ◆ Los servicios web serán desarrollados cumpliendo con los estándares definidos por la W3C y OASIS como: XML como formato estándar para los datos que se vayan a intercambiar, SOAP como protocolo para el intercambio de datos, WSDL para la descripción de la interfaz pública de los servicios web, WS-Security para la autenticación de los actores y la confidencialidad de los mensajes enviados y finalmente la UDDI para publicar la información de los servicios web y comprobar qué servicios web están disponibles, la influencia de estos RNF recaen directamente sobre la capa de negocios la cuál presta estos servicios directamente.
- ◆ El sistema podrá ser escalado fácilmente de forma vertical mejorando los requerimientos de hardware de los servidores, por ejemplo: aumentando la memoria RAM, cambiando el microprocesador por otro de mayor rendimiento, etcétera, y horizontalmente mediante balances de carga a través de clústeres de servidores en la medida que crezca la demanda.

#### ❖ **Restricciones de acuerdo a la estrategia de diseño**

- ◆ El diseño de las aplicaciones se hará utilizando la Programación Orientada a Objetos (POO). Encapsulación de la lógica por clases.
- ◆ Se utilizará las ventajas de tecnología que brinda el framework Axis2 como motor de servicios web.

- ◆ Se utilizará las tecnologías que brindan los frameworks definidos para cada una de las capas de la aplicación:
  - Para la capa de presentación: *MyFaces*, aplicación Web.
  - Para la capa de lógica del negocio: los objetos del negocio, framework *Spring*, la Inversión de Control (*IoC: Inversion of Control*), la programación Orientada a Aspectos y el módulo Acegi de Spring para la seguridad.
  - Para la capa de Acceso a Datos: framework *Hibernate*.

#### ❖ Herramientas de desarrollo

- ◆ Para el modelado Visual Paradigm el cual demanda como mínimo 256 MB de RAM y recomendado 512 MB de RAM.
- ◆ Para implementación se empleará como IDE de desarrollo *Eclipse 3.3.1* el cual demanda como mínimo 768 MB de RAM y recomendado 1.0 GB de RAM al integrarlo con los plug-ins necesarios y un procesador Intel Pentium IV a 2.4 GHz o superior.
- ◆ Para implementación los plug-ins serán:
  - *JBossTools* y *WTP-SDK*, para desarrollar la capa de presentación y los servicios web.
  - *Spring-ide*, para implementar el Framework Spring 2.0.
  - *Hibernate Tools*, el cual facilita la ingeniería inversa y reversa de los ficheros de mapeo entre clases java y tablas de Base de Datos.
  - *JUnit* el cual facilita las pruebas de unidad.
  - *SVNeclipse* el cual facilita la integración con Subversión
- ◆ Como servidor de Base Datos PostgreSQL. (512 MB de RAM).
- ◆ Como servidor de control de versiones Subversión.

### 2.2.4. Tamaño y Rendimiento

En esta sección se realiza una descripción de las características y dimensiones importantes del software que afectan directamente a la arquitectura propuesta, así como también algunas restricciones de rendimiento para la puesta en marcha del Sistema de Gestión de Inventario.

#### ◆ Crecimiento de los datos

El Sistema Gestor de Base de Datos seleccionado para la propuesta arquitectónica es PostgreSQL el cual presenta entre sus características: juegos de caracteres internacionales, tipos de codificación de caracteres multi-byte, Unicode, y es consciente de localización para su clasificación, caso de sensibilidad, y de formato. Es altamente escalable, tanto en la enorme



cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede tolerar. (PostgreSQL, 2008)

En la siguiente tabla se muestran algunas limitantes de PostgreSQL:

Límite	Valor
Tamaño máximo de la Base de Datos	Ilimitada
Tamaño máximo de tabla	32 TB
Tamaño máximo de fila	1.6 TB
Tamaño máximo de campo	1 GB
Máximo de filas por tabla	Ilimitada
Máximo de columnas por tabla	250 - 1600 depende del tipo de dato de las columnas
Máximo de llaves por tabla	Ilimitada

Tabla 1. Potencialidades de PostgreSQL (PostgreSQL, 2008)

Independientemente de la capacidad de almacenamiento y potencialidades de PostgreSQL se hace necesario analizar el crecimiento de los datos que manipulará el sistema. Para hacer esta estimación se utilizará como referencia la base de datos del Sistema de Inventario de la empresa *Cubalse* (una de las empresas más grandes del país que trabaja con el clasificador de productos nacional y bajo las actuales regulaciones del Ministerios de Finanzas y Precios), en la cual se observó que en peor de los casos se hacían alrededor de las 100 operaciones diarias. A partir de este dato se estima entonces que en las peores situaciones la base de datos del Sistema de Gestión de Inventario, que cuenta con 60 tablas persistentes, crezca unos 24 MB diario. Para más información (anexo 16).

#### ◆ Tiempo de respuesta

PostgreSQL contribuye con los tiempos de respuesta del sistema puesto a que presenta una serie de funcionalidades como las que se presentan a continuación que favorecen el rendimiento de las transacciones a la base de datos:

- Recorridos de Mapas de Bits: los índices son convertidos a mapas de bits en memoria cuando es apropiado, otorgando hasta veinte veces más rendimiento en consultas complejas para tablas muy grandes. Esto también ayuda a simplificar la administración de bases de datos reduciendo significativamente la necesidad de índices multi-columna. (PostgreSQL, 2008).

- Particionamiento de Tablas: El optimizador de consultas es capaz de evitar recorrer secciones completas de tablas grandes, a través de una técnica conocida como Exclusión por Restricciones. Similar a las características de Particionado de Tablas de otros sistemas gestores de datos, esta característica mejora tanto el rendimiento como la gestión de datos para tablas de varios gigabytes. (PostgreSQL, 2008).
- Bloqueos Compartidos de Registros: El modelo de bloqueos, mejor que a nivel de registro, de PostgreSQL soporta niveles de concurrencia aún mayores que las versiones anteriores, a través de la adición de candados compartidos a nivel de registro para llaves foráneas. Estos candados compartidos mejoran el rendimiento de inserción y actualización para muchas aplicaciones Procesamiento de Transacciones En Línea (OLTP: OnLine Transaction Processing) de gran concurrencia. (PostgreSQL, 2008).

También gran parte de los tiempos de respuesta del sistema serán manejados por el servidor de aplicaciones Apache el cual implementará un balance de carga entre un clúster de servidores web (*Tomcat*) donde estarán desplegados los servicios que contienen toda la lógica de negocio de la aplicación. Esto posibilitará un mejor rendimiento por parte de los servicios web al estar desplegados en varios servidores influyendo también en los tiempos de respuesta del Sistema de Gestión de Inventarios.

#### ◆ Niveles Concurrencia

Los niveles de concurrencia del sistema serán manejados desde el Sistema Gestor de Base de Datos PostgreSQL, el cual maneja la concurrencia mediante un modelo de control de concurrencia que evita el bloqueo de los lectores ante acciones de los escritores y viceversa.

#### ***Multiversion Concurrency Control (MVCC)***

Es una tecnología que PostgreSQL usa para evitar bloqueos innecesarios. MVCC está considerado mejor que el bloqueo a nivel de fila porque un lector nunca es bloqueado por un escritor. En su lugar, PostgreSQL mantiene una ruta a todas las transacciones realizadas por los usuarios de la base de datos. PostgreSQL es capaz entonces de manejar los registros sin necesidad de que los usuarios tengan que esperar a que los registros estén disponibles.

A diferencia de los sistemas de base de datos tradicionales que usan bloqueos para el control de la concurrencia de dato, PostgreSQL mantiene la concurrencia de datos mediante el uso de un modelo de multi-versión (*Multiversion Concurrency Control, MVCC*). Esto significa que mientras consultas la base de datos cada transacción ve una versión de la base de datos, tal y

como lo fue unos instantes antes, sin tener en cuenta el estado actual de los datos subyacentes. De esta forma se evita la visualización de datos inconsistentes que pueden ser a causa de otra operación de actualización, de una transacción concurrente sobre la misma fila de datos, proporcionando transacciones aisladas para cada consulta a la base de datos. (PostgreSQL, 2005)

La principal ventaja de utilizar el modelo de control de concurrencia MVCC, en lugar de bloqueo, es que en MVCC los bloqueos adquiridos para consultas de lecturas de datos no entran en conflicto con los bloqueos adquiridos para la escritura de datos, por lo que la lectura nunca bloquea a la escritura y viceversa.

Por otra parte la única limitante que puede presentar la propuesta de arquitectura en cuanto al máximo de conexiones concurrentes, es el hardware del cual se disponga para los servidores. Para PostgreSQL el máximo de conexiones es un parámetro configurable que depende, entre otras cosas, de la cantidad de memoria RAM.

También gran parte de los tiempos de respuesta del sistema serán manejados por el servidor de aplicaciones Apache el cual implementará un balance de carga entre un clúster de servidores web (*Tomcat*) donde estarán desplegados los servicios que contienen toda la lógica de negocio de la aplicación. Esto posibilitará al sistema atender un gran número de peticiones concurrentes ante la posibilidad de que el sistema crezca en cantidad de usuarios, facilitando de esta manera también la disponibilidad y la escalabilidad del Sistema de Gestión de Inventarios.

### 2.2.5. Vista de Casos de Uso

Esta vista va a permitir definir los escenarios o casos de uso que serán de interés en cada iteración del ciclo de vida del proyecto. Además va a permitir describir los escenarios o casos de uso que tienen un alto grado de importancia para la arquitectura puesto a que encapsulan gran parte de la funcionalidad central del sistema. A estos escenarios o casos de uso se les llama casos de uso arquitectónicamente significativos porque son aquellos que describen funcionalidades imprescindibles para el sistema, y que a través de ellos se valida la arquitectura propuesta para el mismo.

A continuación se presenta la vista de casos de uso del Módulo de Nomencladores

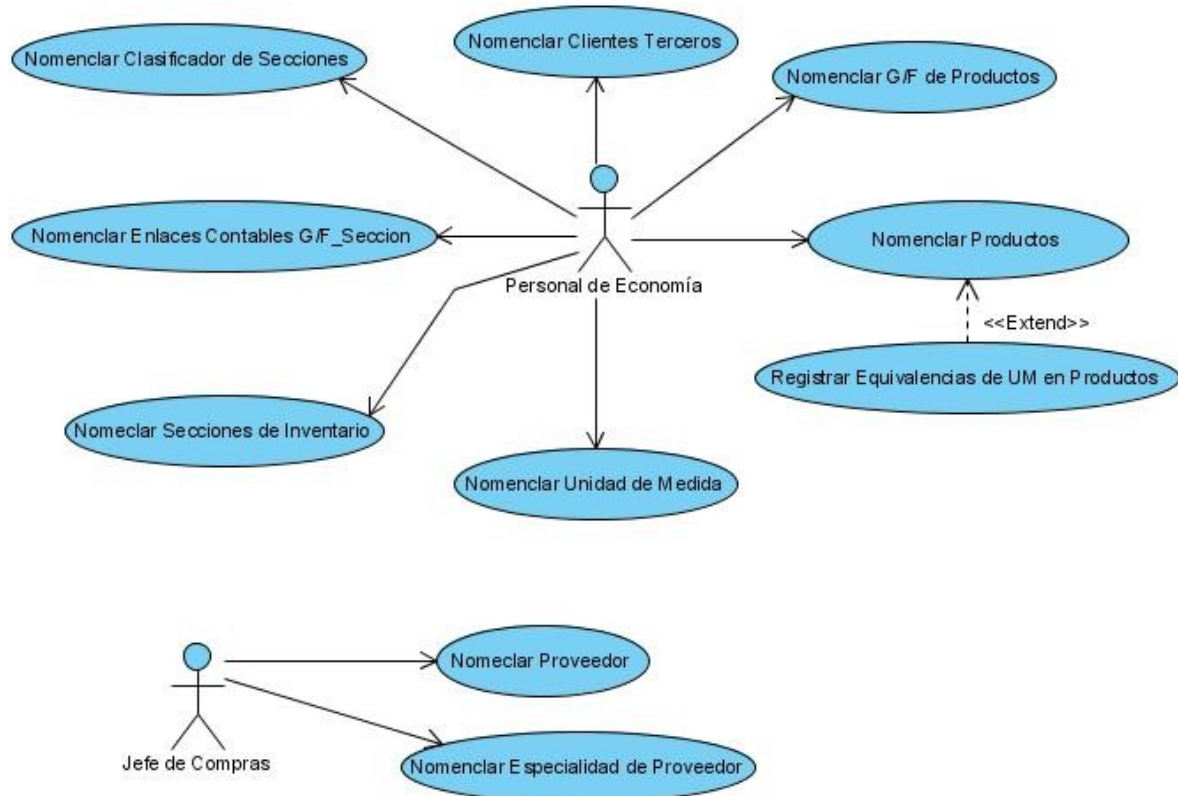


Figura 9. Vista de casos de uso del Módulo Nomencladores

El Módulo de Nomencladores es de suma importancia para el funcionamiento del sistema puesto a constituye la base conceptual sobre la cual se realizarán las distintas operaciones de negocio. Debido a esto todos los casos de usos del diagrama de casos de uso del sub-módulo de Nomencladores se consideran arquitectónicamente significativos.

Como se pudo observar anteriormente en el Sistema de Gestión de Inventarios se identificaron siete sub-módulos. En esta sección sólo se muestra la vista de casos de uso para el Módulo Nomencladores, el cual presenta los siguientes casos de uso:

- ◆ **Nomenclar Clasificación de Secciones:** Permite definir las clasificaciones de secciones y las características que deben cumplir.
- ◆ **Nomenclar Clientes Terceros:** Permite definir, modificar y eliminar los clientes.
- ◆ **Nomenclar Grupo/Familia de Productos:** posibilita registrar, modificar y eliminar grupos de productos y sus respectivas familias (si no tienen dependencias).
- ◆ **Nomenclar Productos:** permite registrar, modificar y eliminar los productos.
- ◆ **Registrar Equivalencias de Unidad de Medida en Productos:** permite registrar las equivalencias entre las unidades de medidas en que pueden encontrarse un producto.

- ◆ **Nomenclar Unidad de Medida:** permite registrar, modificar y eliminar una unidades de medidas de un producto con las que trabaja la entidad, así como establecer las equivalencias entre ellas.
- ◆ **Nomenclar Secciones de Inventario:** permite registrar, modificar y eliminar las secciones de inventario.
- ◆ **Nomenclar Enlaces Contables Grupo/Familia Sección:** Permite registrar, modificar y eliminar los enlaces contables a las cuantas de inventario.
- ◆ **Nomenclar Proveedores:** permite registrar, modificar y eliminar un proveedor.
- ◆ **Nomenclar Especialidad de Proveedores:** Permite registrar, modificar y eliminar especialidades de proveedores, así como relacionarlas a los G/F correspondientes.

## 2.2.6. Vista Lógica

### Visión general de la arquitectura – Alineamiento de paquetes, subsistemas y capaz

En la descripción de la arquitectura, la vista lógica describe las clases más importantes que formarán parte del ciclo de desarrollo atendiendo a los niveles de abstracción. Aquí se describen los paquetes utilizados, y las relaciones que entre ellos existen ya sea de dependencia o de uso.

A continuación se ilustra la división en módulos del Sistema de Gestión de Inventarios con el objetivo de hacer el sistema más reusable y facilitar el mantenimiento ya que cualquier cambio en los procesos del negocio se tendría que cambiar solo en el módulo al que pertenece la funcionalidad que debe cambiar. Facilita también el trabajo del equipo de desarrollo ya que permite que se puedan ir desarrollando módulos paralelamente y saber cual es la dependencia entre los mismos.

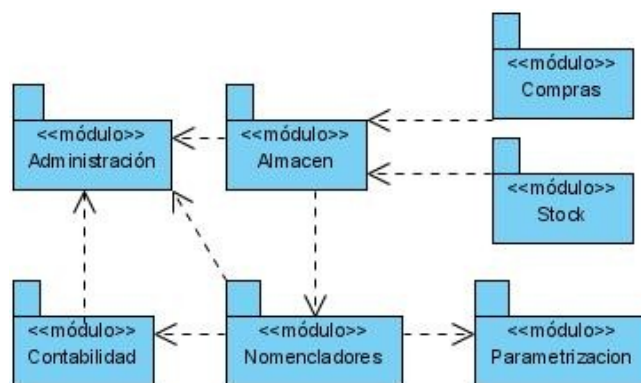


Figura 10. Módulos del Sistema de Gestión de Inventarios

Los módulos principales identificados son:

- ◆ **Administración:** Contiene la realización de los Casos de Uso correspondiente con la Gestión de Usuarios, Roles, Terminales.

- ◆ **Parametrización:** Contiene la realización de los Casos de Usos correspondiente con la configuración de las estructura de la organización, la configuración de los comprobantes contables, dígitos de los códigos y otros.
- ◆ **Nomencladores:** Contiene la realización de los Casos de Uso de nombrar producto, sección, unidad de medida y otros.
- ◆ **Almacén/Inventario:** Contiene la realización de los Casos de Uso de las principales operaciones que se realizan en el sistema, por ejemplo: entrar productos por compra, solicitar productos, rebajar productos por movimientos y otras.
- ◆ **Stock:** Contiene la realización de los Casos de Uso que permiten la gestión de Stock de la organización.
- ◆ **Compras:** Contiene la realización de los Casos de Uso que permitan la gestión de las compras y los procesos que de ella se derivan.
- ◆ **Contabilidad:** Contiene la realización de los Casos de Uso que permita configurar los parámetros fundamentales para el que sistema pueda realizar operaciones como la Validación Contable, y la generación de los comprobantes contables.

La distribución de la aplicación se hará de la siguiente manera:

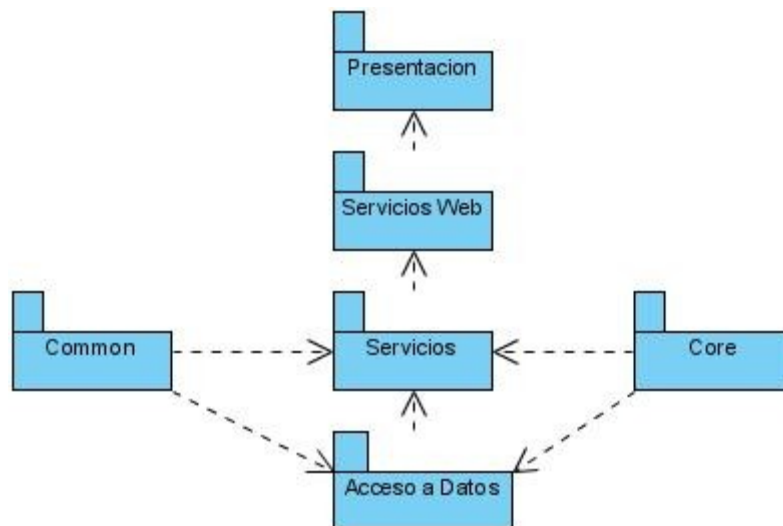


Figura 11. Estructura de Capas del Sistema

- ◆ **Presentación:** Clases y componentes del framework MyFaces.
- ◆ **Servicios Web:** Servicios Web que encapsulan la lógica del negocio de los servicios o subprocesos del sistema inventario, componentes del framework Axis2.
- ◆ **Servicios:** Clases que controlan la Lógica del Negocio, componentes del framework Spring

- ◆ **Acceso a Datos:** Clases que controlan la lógica transaccional y la interacción con la Base de Datos, componentes del framework Hibernate.
- ◆ **Common:** Objetos de Negocio proporcionados por el mapeo de las clases del framework Hibernate, como estos no guardan información de estado que los identifican como objetos de persistencia se pueden utilizar desde cualquier lado de la aplicación.
- ◆ **Core:** Paquete que encierra clases comunes, componentes comunes para todas las capas de la aplicación, contiene la configuración de los frameworks.

Los principales subsistemas y las interfaces que permiten la comunicación entre los componentes son:

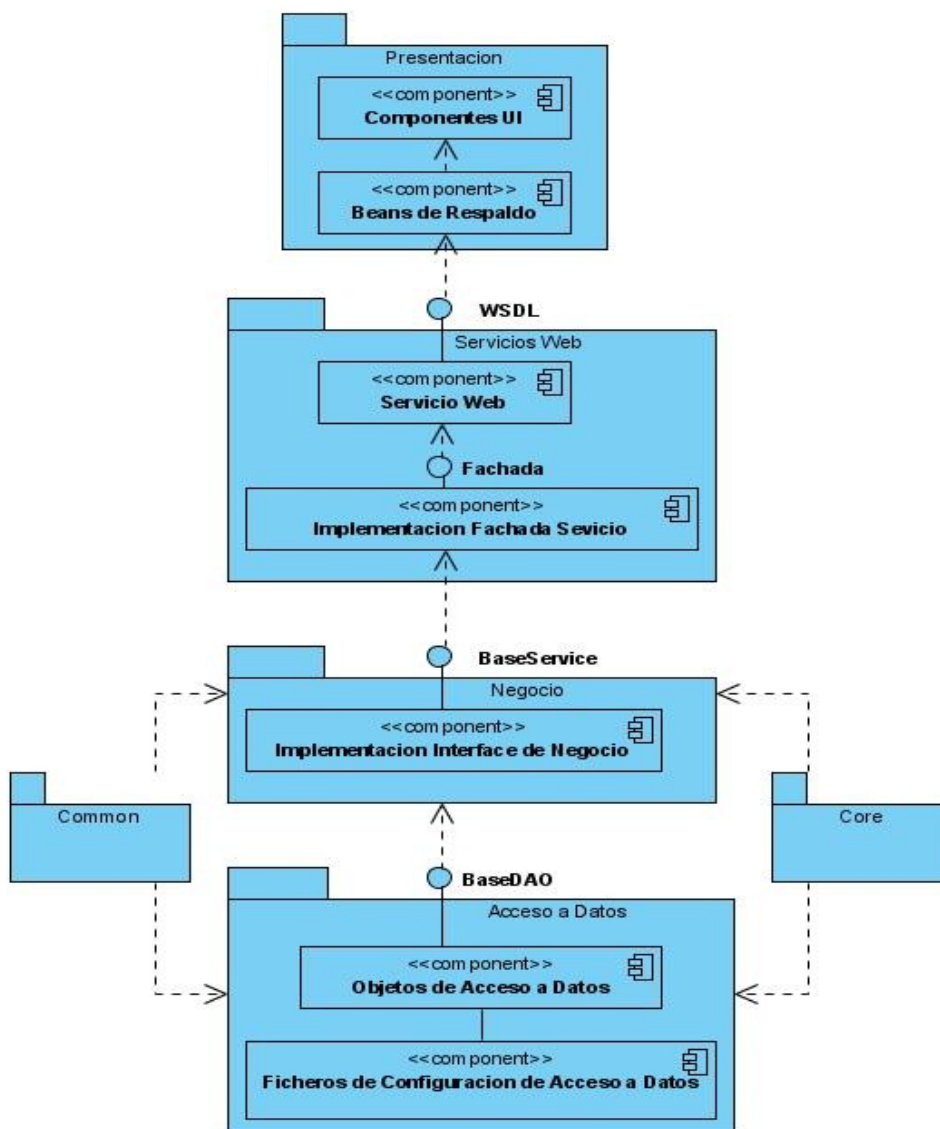


Figura 12. Subsistemas de diseño de la aplicación

## Paquete Core

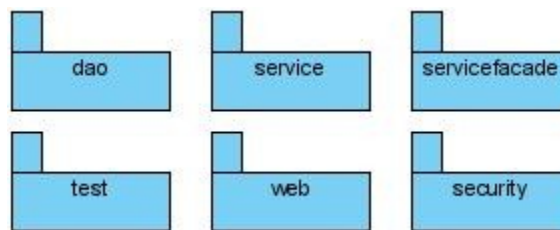


Figura 13. Paquetes que componen al Core.

- ◆ El paquete **dao** contiene la implementación de la interfaz *BaseDao*: clases que controlan el acceso a datos, contiene la funcionalidad necesaria para acceder a los datos del repositorio de datos.
- ◆ El paquete **security** contiene las clases que implementan la seguridad con el framework Acegi.
- ◆ Paquete **service** contiene la declaración e implementación de la interfaz *BaseService*: clases que controlan la lógica de negocio de la aplicación, ejecutan transacciones, cálculos.
- ◆ El paquete **test** contiene la declaración e implementaciones de la interfaz para usar en las pruebas de *JUnit BaseTestCase*.
- ◆ El paquete **servicefacade** contiene los objetos de servicios a utilizar en la fachada de negocio.
- ◆ El paquete **web** contiene la implementación de *BaseBean*, clases de utilidad de JSF y reglas de navegación.

## Paquete Common

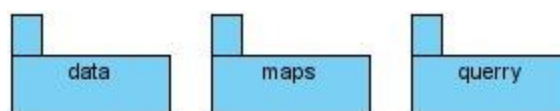


Figura 14. Paquetes de Common

El paquete contiene las clases y objetos del mapeo, los ficheros XML del mapeo y los ficheros para las consultas.

- ◆ El paquete **data** contiene los Objetos de Negocio que se generan a partir del mapeo de las clases de las tablas de la base de datos. Son clases entidades de Java que contiene los atributos y métodos para acceder a ellos, estos constituyen los objetos que se utilizaran en el sistema.
- ◆ El paquete **maps** contiene los ficheros XML de mapeo, son ficheros que contienen la información de la Base de Datos a la que hace referencia, contiene los campos de cada una de las tablas y sus relaciones.



- ◆ El paquete **query** contiene los ficheros para realizar las consultas a la base de datos.

### Capa de Presentación

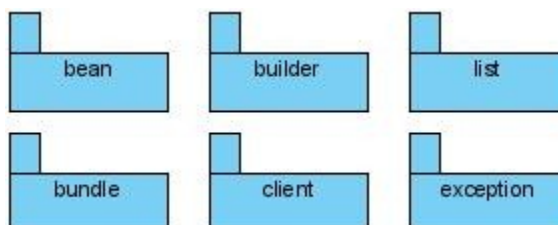


Figura 15. Paquetes de la Capa de Presentación

Paquete donde se incluyen las clases utilizadas por la aplicación que capturan las acciones de la interacción con el de usuario, dando acceso así al sistema, y se establece la configuración de los beans de las paginas JSP.

- ◆ El paquete **bean** contiene las clases beans de respaldo de las paginas MyFaces.
- ◆ El paquete **builder** contiene las clases que permiten convertir a los objetos de un tipo a otro según corresponda en cada caso.
- ◆ El paquete **bundle** contiene ficheros de configuración.
- ◆ El paquete **client** contiene las clases que implementan la conexión con los servicios web, así como los métodos que se necesiten de los mismos.
- ◆ El paquete **exception** contiene clases para el manejo de las excepciones.
- ◆ El paquete **list** contiene las clases que contienen listas de beans.

### Capa de Servicios Web

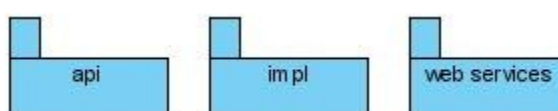


Figura 16. Paquetes de la Capa de Servicios Web

Esta capa lógica contiene las interfaces de las fachadas de los servicios de negocio, su respectiva implementación y los servicios web. Siendo esta fachada la que se exporte como un servicio web y no el propio servicio de negocio, logrando así un mayor encapsulamiento pues se abstrae a quien utilice dicho servicio de cómo se desarrolló. Estos servicios web serán brindados tanto a la capa de presentación del Sistema de Gestión de Inventarios como a cualquier otro servicio o aplicación a través de la UDDI quien expondrá sus contratos (WSDL), y cómo acceder a ellos.

- ◆ El paquete **api** contiene la definición de las clases interfaces de las fachadas.
- ◆ El paquete **impl** contiene la implementación de las interfaces.

- ◆ El paquete **web services** contiene los servicios web desarrollados.

### Capa de Servicios

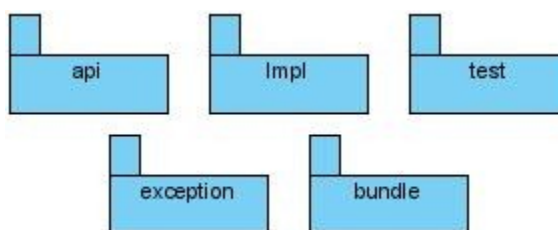


Figura 17. Paquetes de la Capa de Servicios

- ◆ Paquete **api**: Contiene las interfaces de los servicios que hagan falta para la implementación de alguna funcionalidad.
- ◆ Paquete **impl**: Contiene las implementaciones que se quieran hacer de las clases interfaces declaradas.
- ◆ Paquete **test**: Contiene las pruebas de unidad (*JUnit*) para cada uno de los servicios implementados.
- ◆ Paquete **exception**: Contiene las clases para el manejo de las excepciones.
- ◆ Paquete **bundle**: Contiene ficheros de configuración.

### Capa Acceso a datos

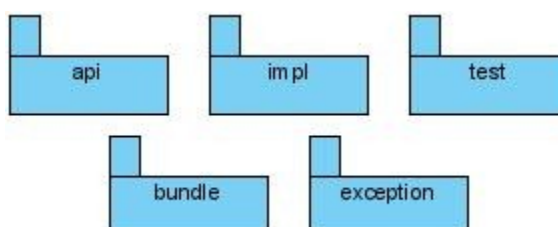


Figura 18. Paquetes de la Capa de Acceso a Datos

- ◆ El paquete **api** contiene las interfaces de los servicios que hagan falta para la implementación de alguna funcionalidad.
- ◆ El paquete **impl** contiene las implementaciones que se quieran hacer de las clases interfaces declaradas.
- ◆ El paquete **test** contiene las pruebas de unidad (*JUnit*) para cada uno de los servicios implementados.
- ◆ Paquete **exception**: Contiene las clases para el manejo de las excepciones.

- ◆ Paquete **bundle**: Contiene los ficheros de configuración de acceso a la base de datos “.properties”.

### Elementos del modelo arquitectónicamente significantes

Para el Sistema de Gestión de Inventarios algunos de los elementos arquitectónicamente significativos son: *BaseDAO*, *BaseService* y la fachada para cada uno de los servicios de negocio que se deseen exportar como un servicio web. A continuación se muestran estas clases con sus respectivos métodos, y en el caso específico de la fachada, la imagen muestra un ejemplo de cómo podría quedar la misma para el caso de la fachada *GrupoWs*, que depende de *BaseService* porque en este caso en particular no se necesita realizar ninguna operación de negocio y las funcionalidades que le brinda son suficientes para realizar las que se precisan. En caso de no ocurrir así pues se tendría una clase que heredaría de *BaseService* todas sus funcionalidades e incluiría aquellas propias que necesite para realizar su lógica de negocio, y ahora en este caso, la fachada dependería de esta nueva clase.

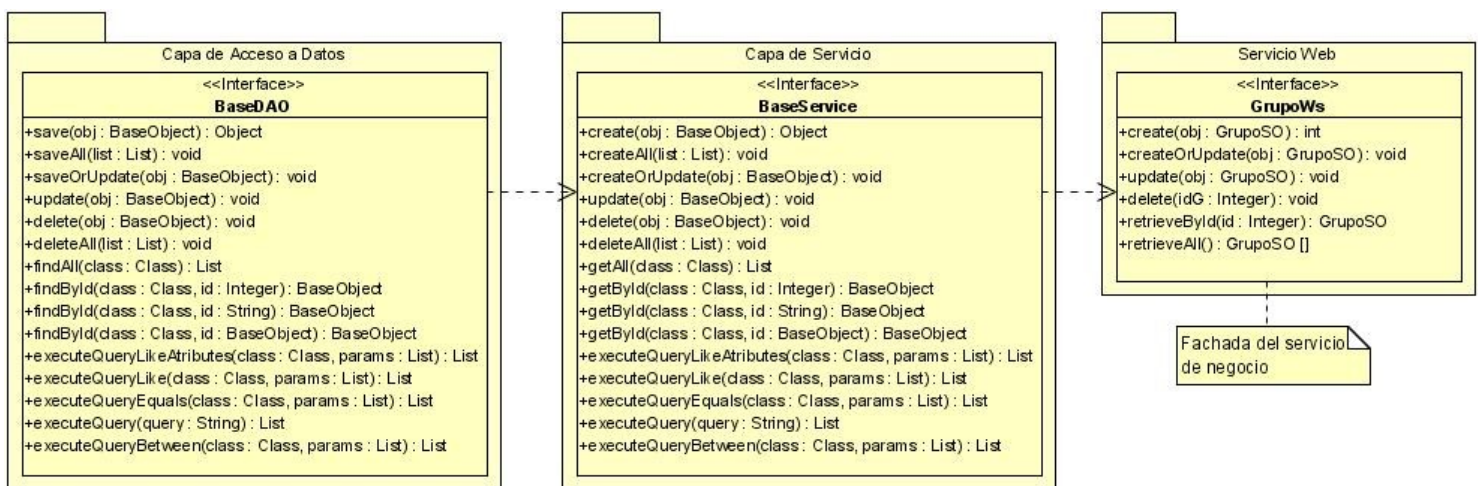


Figura 19. Clases arquitectónicamente significativas.

## 2.2.7. Vista de Despliegue

### Diagrama de Despliegue

Esta vista da una medida de la tecnología necesaria para el correcto funcionamiento del sistema a desarrollar, propone la distribución física de los elementos que lo conforman ya que representa cómo estarán distribuidos y cómo se satisfacen los requerimientos no funcionales de hardware y software.

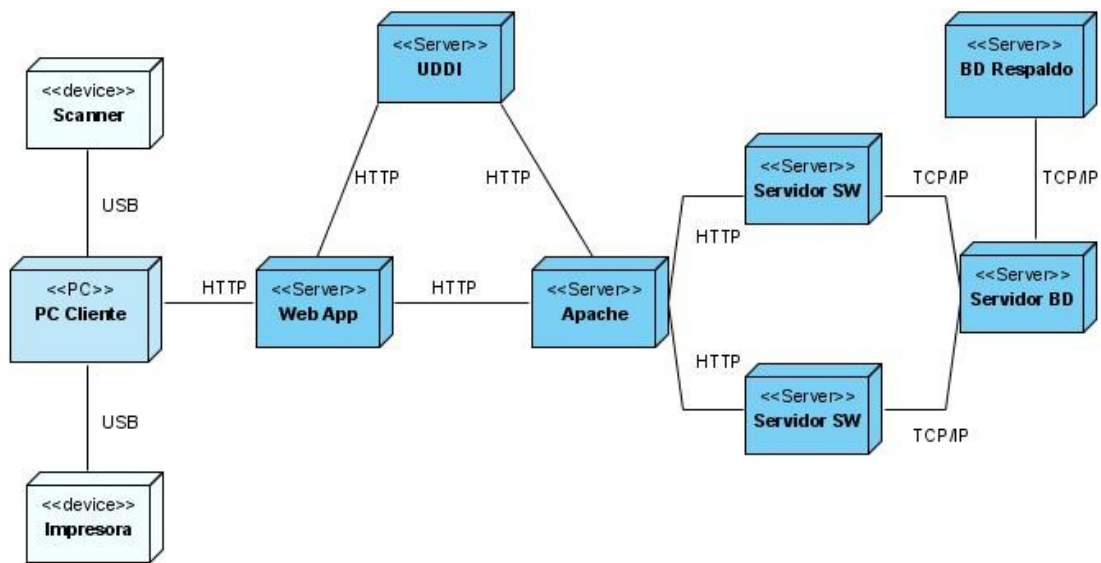


Figura 20. Diagrama de despliegue del Sistema de Gestión de Inventario

#### Nombre de dispositivo: descripción de la capacidad que el dispositivo provee al sistema

- ❖ **Scanner:** Este dispositivo satisface las necesidades de los clientes ante la posibilidad de escanear contratos de proveedores o cualquier otro documento o factura de interés para su resguardo.
- ❖ **Impresora:** Este dispositivo satisface las necesidades de los clientes ante la posibilidad de imprimir reportes obtenidos de la aplicación como constancia de una operación realizada.

#### Nombre del procesador: descripción de la funcionalidad y capacidad del nodo

- ❖ **PC Cliente:** Su función es acceder al sistema e interactuar con el mismo según sus necesidades. Además de interactuar con los dispositivos de escaneo e impresión. Al estar la aplicación desarrollada sobre la web la máquina cliente necesita disponer de muy pocas prestaciones puesto a que sólo necesita un navegador web para poder acceder al sistema y realizar las operaciones necesarias.
- ❖ **Aplicación Web:** En este nodo es donde descansa la capa de presentación del sistema, la cual es accedida por las máquinas clientes a través de un navegador web. Para poder responder a las peticiones interactúa con la UDDI para que le provea la dirección donde se encuentran los servicios necesarios para poder atenderlas.
- ❖ **Servidor Apache:** Este servidor es el encargado de atender las solicitudes, y ante una petición, asignar al servidor que le corresponde dicha petición, realizando de esta forma un balance de carga entre los servidores que contienen los servicios web permitiendo de esta manera adaptarse al creciente número de usuarios y sobrellevar las cargas de trabajo

- ❖ **Servidor SW:** Estos servidores contienen toda la funcionalidad del sistema expuestas mediante servicios web. Son los encargados de publicar dichos servicios para que tanto el propio sistema como cualquier otra aplicación o servicio, pueda descubrirlos mediante la UDDI y así utilizarlos para funcionamiento.
- ❖ **UDDI:** Este nodo constituye el repositorio de los servicios expuestos por el sistema, en otras palabras, constituyen el directorio que se consulta para saber si la entidad brinda un determinado servicio o no, y en que caso afirmativo le proporciona la vía y el cómo acceder a él.
- ❖ **Servidor de Base de Datos:** Es el encargado de almacenar toda la información generada del sistema y replicarla al servidor de respaldo previendo la disponibilidad y la tolerancia a fallos del sistema ante cualquier falla posible.
- ❖ **BD de Respaldo:** Su función es la de almacenar la réplica de información proveniente del Servidor de Base de Datos y ante alguna falla de este prestar servicios por él hasta que se restablezca.

Para conocer detalles sobre la capacidad del nodo remitirse a la sección de objetivos y restricciones arquitectónicas.

## Descripción de elementos e interfaces de comunicación

### <<Nombre tipo de conexión>>: Características físicas de la conexión

- ❖ <<HTTP>>: El Protocolo de Transferencia de Hipertexto es un protocolo define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones, sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores y sigue el esquema petición-respuesta entre un cliente y un servidor.
- ❖ <<HTTPS>>: El Protocolo Seguro de Transferencia de Hipertexto es un protocolo de red basado en HTTP, destinado a la transferencia segura de datos de hipertexto, en otras palabras, es la versión segura de HTTP.
- ❖ <<USB>>: El Bus Universal en Serie es un puerto que sirve para conectar periféricos a una computadora. El estándar incluye la transmisión de energía eléctrica al dispositivo conectado. Para dispositivos multimedia como escáneres y cámaras digitales, el USB se ha convertido en el método estándar de conexión en los últimos años. Para impresoras, el USB ha crecido tanto en popularidad que ha empezado a desplazar a los puertos paralelos porque el USB hace sencillo el poder agregar más de una impresora a una computadora personal.
- ❖ <<TCP/IP>>: La familia de protocolos de Internet es un conjunto de protocolos de red en la que se basa Internet y que permite la transmisión de datos entre redes de computadoras. Denominada en

muchas ocasiones conjunto de protocolos TCP/IP en referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (*TCP*) y Protocolo de Internet (*IP*), que fueron los dos primeros en definirse, y que son los más utilizados de la familia. El TCP/IP es la base de Internet, y sirve para enlazar computadoras que utilizan diferentes Sistemas Operativos, incluyendo computadoras personales, minicomputadoras y computadoras centrales sobre redes de área local (LAN) y área extensa (WAN).

### 2.2.8. Vista de Implementación

Esta vista proporciona una descripción del sistema en función de sus principales capas, subsistemas y componentes. A continuación se muestran los principales paquetes de la aplicación comunes a los módulos identificados para el desarrollo de la misma.

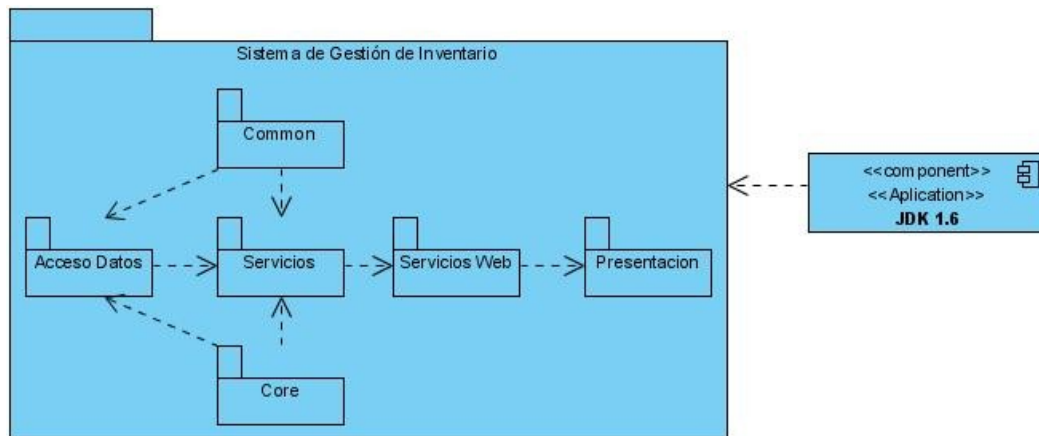


Figura 21. Vista general del Modelo de Implementación

Cada uno de dichos paquetes encapsulan uno o varios componentes que se interrelacionan entre ellos para contribuir a la solución de los elementos de la aplicación y todos estos elementos dependen de la JDK que debe de estar instalada donde este la aplicación para su funcionamiento.

Los componentes están distribuidos en las capas y paquetes de la aplicación de la siguiente forma:

- ❖ **Core:** Contiene los componentes fundamentales para el trabajo en cada una de las capas:



Figura 22. Componentes del paquete Core

❖ Common

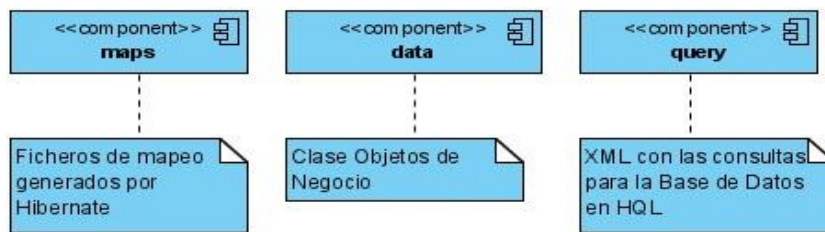


Figura 23. Componentes del paquete Common.

❖ Presentación

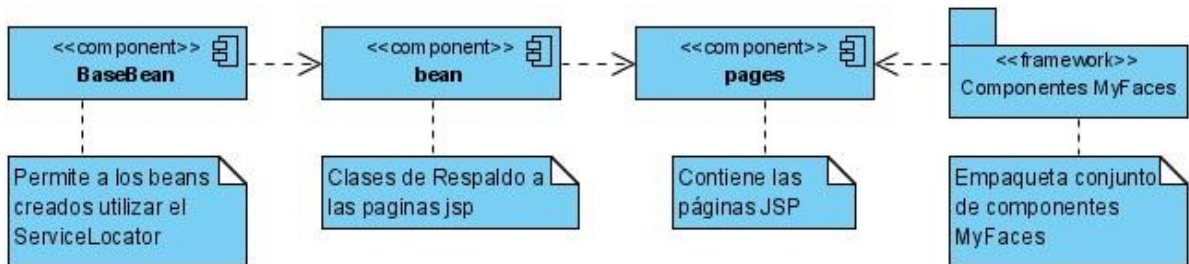


Figura 24. Componentes de la capa de Presentación

❖ Servicios Web

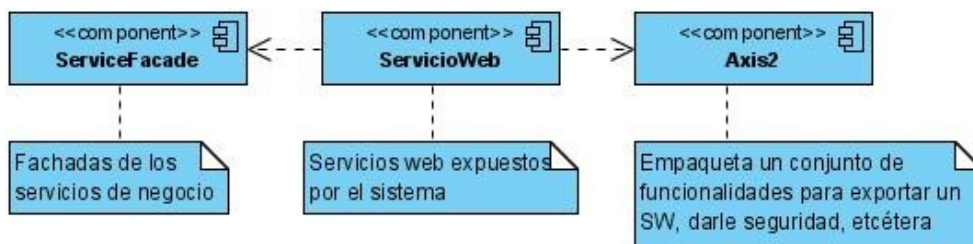


Figura 25. Componentes de los Servicios Web

❖ **Servicios**

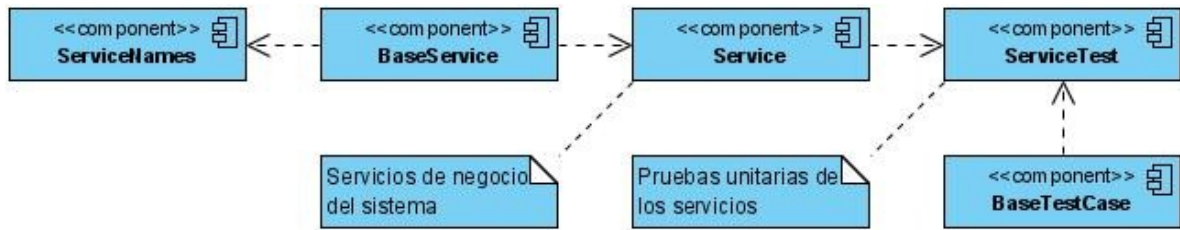


Figura 26. Componentes de los Servicios Web

❖ **Acceso a Datos**

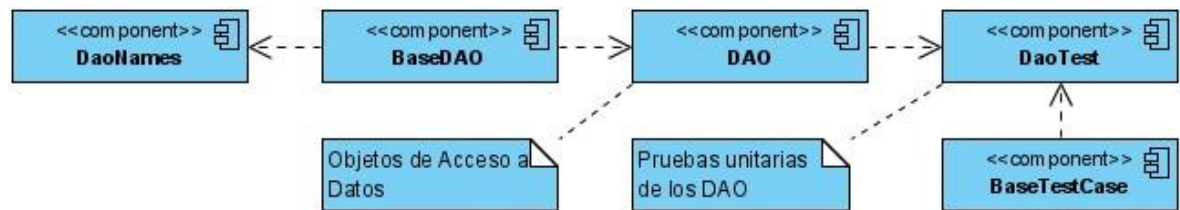


Figura 27. Componentes de los Servicios Web

**2.2.9. Vista de Datos**

Como ya se había hablado anteriormente, la estructuración del sistema en capas permite que se abstraiga la lógica de negocio del almacenamiento de los datos, la capa de acceso a datos contiene toda la lógica de acceso, ya sea: consultas y procedimientos almacenados, dejando a la Base de Datos como simple almacén de datos sin ninguna lógica, solo algunos disparadores para manejar parte de la seguridad en la misma.

En la propuesta arquitectónica para realizar la persistencia de los datos del Sistema de Gestión de Inventario se emplea el *ORM Hibernate* con el Sistema Gestor de Base de Datos *PostgreSQL*.

El modelo de datos del Sistema de Gestión de Inventario cuenta con un total de 60 tablas persistentes que engloban toda la información del sistema. Por cuestiones de espacio sólo se muestran a continuación las tablas relacionadas con los casos de uso mostrados anteriormente en la Vista de Casos de Uso del Módulo de Nomencladores.



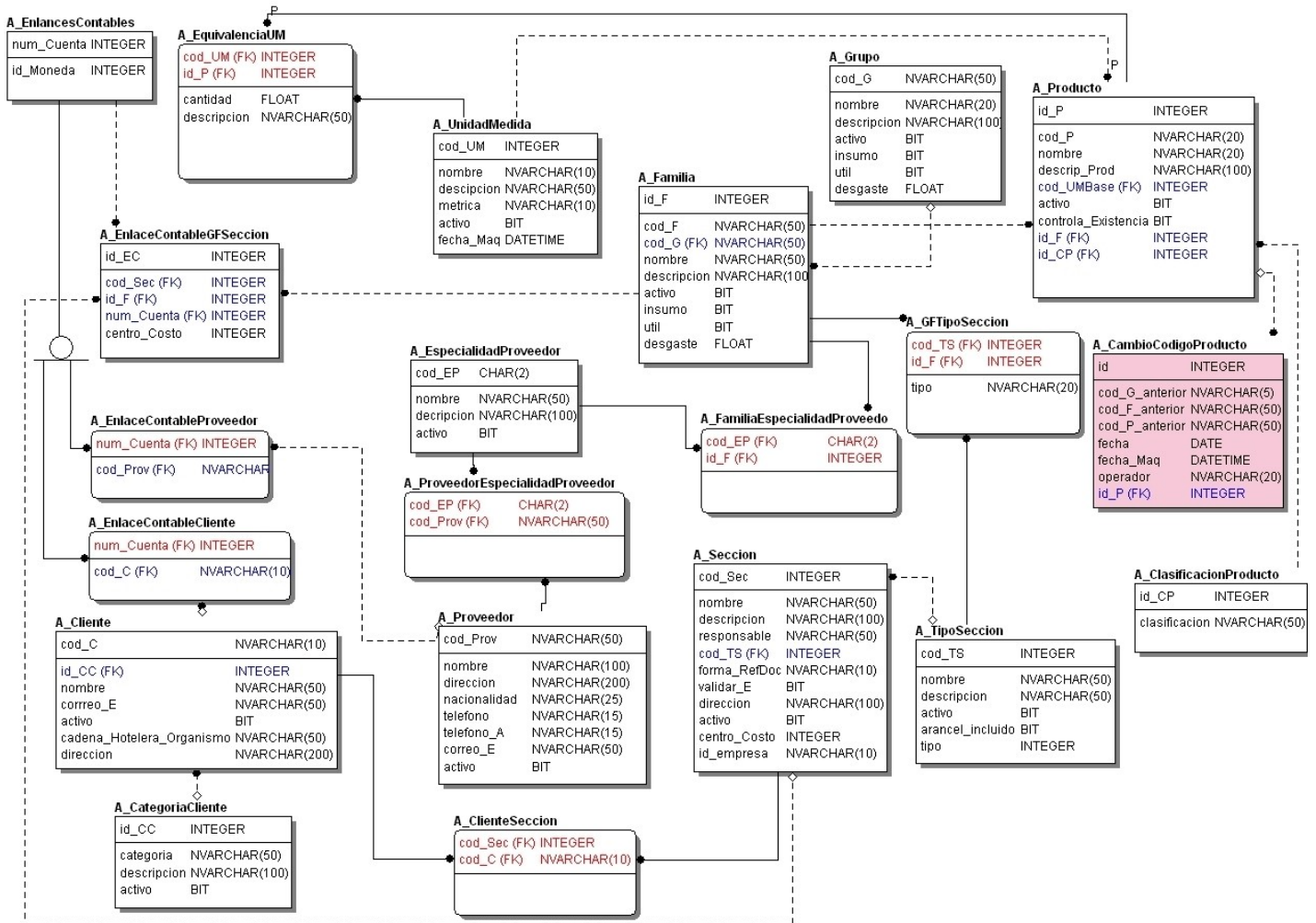


Figura 28. Modelo Entidad Relación del Módulo Nomencladores

### 2.2.10. Flujo de Trabajo

Una vez realizada todas las configuraciones necesarias en eclipse para poder integrar los frameworks a usar, las configuraciones vía XML que referencian a la base de datos que se va a usar y cómo se van a conectar, se puede proceder a crear un servicio web desde cero para que sea consumido posteriormente por el sistema, otro servicio o aplicación.

#### Componentes de la Capa Acceso a datos

Los principales componentes de la capa de acceso a datos son:

- ◆ Fichero de Mapeo (XML Mapping)
- ◆ Objetos de Negocio (BaseObject)
- ◆ Objetos de Acceso a Datos.

Para generar estos componentes:

1. Generar tanto los Ficheros de Mapeo como los Objetos de Negocio, estos se generan automáticamente aprovechando las ventajas que proporciona el entorno de desarrollo eclipse.
2. Una vez que se tienen estos dos componentes se puede proceder a crear los Objetos de Acceso a Datos (DAO), que son los principales componentes de la capa de acceso a datos, los cuales contienen la funcionalidad necesaria para acceder a la Base de Datos, realizar las operaciones y transacciones.
  - a. Crear la clase interfaz *BaseDAO*, la cual va a contener todas las funcionalidades comunes y necesarias para realizar operaciones y transacciones sobre la base de datos. Esta interfaz se crea genérica para que pueda ser utilizada por todos los servicios que se implementen.
  - b. Crear la clase que implementa la interfaz genérica *BaseDAO*. Esta clase hereda de la *HibernateDaoSupport* que es la que le da el soporte para la realizar las operaciones sobre la base de datos y las transacciones.
  - c. Referenciar el nuevo DAO en el fichero de configuración: "*applicationContext-Dao.xml*" el cual a su vez referencia al *bean* "*sesiónFactory* " que es el encargado de crear las sesiones a la base de datos, y en el *DAONames* lugar donde están referenciados todos los objetos de acceso a datos creados.
  - d. Por último probar que todos los métodos del DAO creado funcionan correctamente mediante las pruebas unitarias con *JUnit*.

### Componentes de la Capa de Servicio

Los servicios son los principales componentes de esta capa, los mismos encapsulan funcionalidades del negocio de la aplicación y ejecutan las transacciones, cálculos y devuelven la información que se necesita.

1. Crear la clase interfaz del servicio que se desea desarrollar. Esta interfaz va a heredar de la interfaz *BaseService*, por tanto va a contener todas las funcionalidades de esta interfaz y sólo se le añadirán aquellas propias que necesite.
2. Crear la implementación de la interfaz del servicio. Esta clase implementa la interfaz que hereda de *BaseService*, contendrá una instancia de *BaseDAO* para dar respuesta a sus funcionalidades, y se le inyectará mediante la inversión de control la referencia a la implementación del DAO correspondiente, mediante el constructor o propiedad, a través del XML: *applicationContext-Service.xml*, donde se registra la entrada del nuevo servicio creado al

igual que en el *ServiceNames*, que similar al *DAONames*, contiene el nombre de todos los servicios creados hasta el momento.

3. Finalmente crear la prueba de unidad con *JUnit* para verificar que todos los métodos del servicio funcionan correctamente.

### Componentes de la Fachada del Negocio

El próximo paso es crear la fachada del negocio para continuar con el flujo de trabajo y de esta forma poder terminar de desplegar un servicio web con Axis2. Crear esta fachada tiene como objetivo diferenciar los objetos de negocio de los objetos que necesitan los servicios, ya que es una mala práctica usar los mismos objetos para las dos cosas, debido a que, quien quiera que utilice el servicio no tiene por qué conocer más allá de la información necesaria para utilizar dicho servicio. Por eso hay que hacer una fachada del servicio que se desee exportar. Para esto se realizan los siguientes pasos:

1. Crear la clase interfaz con todos los métodos del servicio y en el caso de aquellos métodos que necesiten de objetos de negocio para realizar las operaciones, se le sustituyen por los objetos de servicios correspondientes debido a las razones antes mencionadas. Para hacer esto se deben apoyar en la clase “*Builder*” correspondiente al servicio (se debe crear para cada servicio), esta contendrá las funcionalidades necesarias para hacer los cambios pertinentes.
2. Crear la clase que implementa la interfaz creada. Para esto contendrá como atributo la interfaz del servicio que se quiere exportar y mediante la inyección de dependencia se le especificará que implementación tomar.
3. Referenciar el nuevo *bean* en el fichero de configuración: *applicationContext-WebService.xml* el cual referencia a la implementación del servicio deseado.

### Servicios Web

Para finalizar con la creación del servicio web basta con seguir las especificaciones de Axis2 para exportar un servicio de Spring:

1. Crear una carpeta con el nombre del servicio web dentro del directorio: “*WebContent/WEB-INF/services*”.
2. Crear dentro de la carpeta creada anteriormente otra llamada: *META-INF*.
3. Crear dentro de la carpeta creada anteriormente el archivo “*service.xml*” donde se especifica el nombre del servicio web creado que tiene que coincidir con el nombre de la carpeta creada en el primer paso, una descripción del servicio que no es el *WSDL*, y el *bean* del servicio al cual hará referencia una vez que se invoque al servicio web.

### Componentes de la capa de presentación

Como ya se había explicado anteriormente esta capa es la encargada de gestionar la interacción hombre - sistema. Para desarrollar los componentes necesarios para esto:

1. Desarrollar los objetos del modelo (*beans*) de respaldo de las páginas *MyFaces*: estos son los objetos que contienen los datos.
2. Referenciar las declaraciones de los beans controladores al fichero de configuración de la aplicación (*faces-managed-beans.xml*)
3. Desarrollar la página *JSP* con los componentes y formularios necesarios para realizar la operación necesaria.
4. Definir la navegación entre las páginas en el archivo *faces-config.xml*.

#### 2.2.11. Conclusiones Parciales

La propuesta de solución dada en este capítulo para el Sistema de Gestión de Inventarios abarca parte del artefacto Documento Descripción de la Arquitectura, el cual se complementa en el siguiente capítulo con la especificación de las herramientas, frameworks y el análisis que se realiza de la calidad de la misma. Este artefacto es de construcción obligatoria, y constante refinamiento por parte de los arquitectos de software del sistema. En el mismo se identificaron los frameworks a emplear para el desarrollo de la solución, el estilo y los patrones, así como las 4+1 vistas de Kruchten que brindaron una representación arquitectónica del sistema.

## CAPÍTULO 3. AMBIENTE DE DESARROLLO Y CALIDAD EN LA ARQUITECTURA

### 3.1. Introducción

En este capítulo se realiza la especificación de las herramientas que brindarán soporte a la propuesta arquitectónica realizada en el capítulo anterior para el Sistema de Gestión de Inventario. Además se realiza un análisis de la propuesta de solución desde el punto de vista de la calidad para examinar cómo se le da cumplimiento a las variables definidas en el problema que da pie a este trabajo de diploma. Complementándose de esta forma el artefacto Documento Descripción de la Arquitectura.

### 3.2. Ambiente de Desarrollo

#### 3.2.1. Herramientas Horizontales

##### 3.2.1.1. Sistema Operativo

Detalles:

- ◆ Nombre Completo: **Microsoft Windows**
- ◆ Versión a utilizar: **Windows XP Professional (eXPerience)**
- ◆ *Compañía, Empresa, Grupo que lo creó y brinda soporte: Microsoft*
- ◆ Tipo de Licencia: **Microsoft CLUF (EULA)**
- ◆ URL en Internet: <http://www.microsoft.com>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Producción Facultad 3.
- ◆ Formación y Recursos Humanos Disponibles.
  - Cursos, Manuales, Documentación que garantice el aprendizaje.
 

Existe buena experiencia en el uso de este sistema operativo, y es muy utilizado en la comunidad universitaria. Existen varios cursos optativos, que se han impartido, además, en internet existe muy buena documentación.
- ◆ Información General
  - Requerimientos Mínimos de Hardware:
 

Procesadores i386 y AMD™ 64 que cumplan con las siguientes características:

    - Procesador 233 MHz x86 o superior
    - 128 MB de RAM
    - 1.5 GB de espacio disponible en el disco duro
    - Unidad de CD-ROM o DVD-ROM
    - Adaptador de video y monitor con resolución Super VGA(800x600) o superior

- Tarjeta de sonido.
- Principales Características:
  - Compatibilidad con aplicaciones.
  - Actualizaciones automáticas.
  - Secuencias más rápidas de inicio y de hibernación.
  - Capacidad del sistema operativo de desconectar un dispositivo externo sin necesidad de reiniciar.
  - Soporte para los estándares de hardware más recientes
  - Opciones de inicio del modo a prueba de fallos
  - Seguridad de red con Windows Firewall ofrece soporte de perfil múltiple.
  - Revoluciona el trabajo de los usuarios remotos.
  - Ofrece una nueva interfaz gráfica.
  - Fiable: Protección de archivos de Windows, Seguridad IP (IPSec), etc.
  - Fácil de Usar: Entorno de usuario adaptable,
  - Incluidos varios programas, como son, navegador Internet Explorer, mensajería instantánea Messenger, reproductor Windows Media Player 10, editor de fotografía paint, Edición de video Windows Movie Maker, editor de texto Bloc de notas, Correo electrónico Outlook Express, juegos solitario, busca minas.

### 3.2.1.2. Antivirus

Detalles:

- ◆ Nombre Completo: **Kaspersky Anti-Virus Personal**
- ◆ Versiones a utilizar: Version **6.0**
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: Kaspersky Lab compañía de seguridad en computadoras. <http://www.kaspersky.com/>
- ◆ Tipo de Licencia: **Shareware**
- ◆ URL en Internet: <http://www.kaspersky.com/products.html>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Antivirus establecido en las Políticas de Seguridad en ambiente Windows en la UCI.
- ◆ Formación y Recursos Humanos Disponibles.
  - Cursos, Manuales, Documentación que garantice el aprendizaje.
  - Facilidad de uso.
- ◆ Información General:

- Requerimientos Mínimos de Hardware:
  - Procesador Intel Pentium 300 MHz o superior (o equivalente)
  - 128 MB de RAM
  - 50 MB de espacio disponible en el disco duro
  - CD-ROM (si se instala desde CD)
  - Conexión de Red (para actualizaciones del producto y la base de datos antivirus vía FTP al servidor de disponible \\ucistore.uci.cu)
  - Microsoft Windows Installer 2.0 o superior.
- Principales Características:
  - Tres grados de protección contra las amenazas existentes y nuevas de Internet: 1) Actualización de bases de datos antivirus cada hora, 2) Análisis preliminar de comportamientos sospechosos, 3) Análisis de comportamientos sospechosos en tiempo real.
  - Protección contra virus, programas troyanos y gusanos.
  - Protección contra spyware y adware.
  - Análisis de ficheros, correo y tráfico de Internet en tiempo real.
  - Defensa contra virus durante el uso de ICQ y otros mensajeros instantáneos.
  - Protección contra todo tipo de keyloggers y todos los tipos de rootkits.
  - Actualización automática de las bases de firmas antivirus.
  - Reversión de los cambios indeseables hechos a su ordenador.
  - Autodefensa del antivirus contra deshabilitación o detención.
  - Herramientas para crear un disco de restauración del sistema.

**3.2.1.3. Gestión de Recursos**

Los recursos necesarios para proceder con el desarrollo del Sistema de Gestión de Inventarios se muestran a continuación en la siguiente tabla donde se hace una relación del levantamiento tecnológico y la asignación de recursos por roles.

Laboratorio 19								
Asignado a:	PC	RAM (MB)	UPS	Monitor	Mouse	Teclado	Impresora	Scanner
Líder de proyecto	1	512 MB	1	1	1	1	1	1
Equipo Planificación	2	512 MB	1	1	1	1		
Equipo Arquitectura	3	1024 MB	1	1	1	1		
Equipo Capacitación	2	1024 MB	1	1	1	1		

Equipo Calidad	2	512 MB	1	1	1	1		
Equipo Base de Datos	2	512 MB	1	1	1	1		
Equipo Gestión de Configuración	2	512 MB	1	1	1	1		
Analistas	3	512 MB	1	1	1	1		
Diseñadores	3	512 MB	1	1	1	1		
Desarrolladores	9	1024 MB	1	1	1	1		
Revisores Técnicos	1	512 MB	1	1	1	1		
<b>Total</b>	<b>30</b>	<b>44(512)</b>	<b>40</b>	<b>40</b>	<b>40</b>	<b>40</b>	<b>1</b>	<b>1</b>

Tabla 2. Levantamiento tecnológico y asignación de recursos

### 3.2.1.4. Control de Versiones

Detalles:

- ◆ Nombre Completo: **Subversion**
- ◆ Versiones a utilizar:
  - Servidor: **Subversion 1.4.6**
  - Cliente: **RapidSVN 0.9.6**
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: CollabNet, Inc.
- ◆ Tipo de Licencia: Apache/BSD
- ◆ URL en Internet: <http://www.tigris.org/>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la universidad.
- ◆ Formación y Recursos Humanos Disponibles.
  - Cantidad de Estudiantes y Profesores capacitados.
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Intel Pentium 700 MHz o superior (o equivalente)
    - 128 MB mínimo de RAM
  - Principales Características.
    - Se sigue la historia de los archivos y directorios a través de copias y renombrados.
    - Maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).
    - Se sigue la historia de los archivos y directorios a través de copias y renombrados.
    - Permite selectivamente el bloqueo de archivos. Se usa en archivos binarios que, al no



poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez.

- Varios Clientes: Subclipse, RapidSVN. Para MAC, pueden emplearse los interfaces SvnX, RapidSVN y Zigversion.
- Integración: **Tiene integración con Eclipse mediante un plug-in llamado Subclipse**

### 3.2.1.5. Gestión de Proyecto

Detalles:

- ◆ Nombre Completo: **DotProject**
- ◆ Versiones a utilizar: Version **2.0.4**
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: Esta construido por aplicaciones de Código abierto y es mantenida por un pequeño pero dedicado grupo de voluntarios. Sitio de los creadores: <http://www.dotmarketing.org>
- ◆ Tipo de Licencia: **v2.x es GPL v2**
- ◆ URL en Internet: <http://www.dotproject.net>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la Universidad.
- ◆ Formación y Recursos Humanos Disponibles:
  - Cursos, Manuales, Documentación que garantice el aprendizaje.
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Intel Pentium 300 MHz o superior (o equivalente)
  - Principales Características:
    - Aplicación web desarrollado en PHP y MySQL, permite la administración de usuarios, sistemas de tickets por email, administración de clientes y empresas, listado de proyectos, listado de tareas, lista de contactos, calendario.
    - Multiplataforma.
    - Los módulos con los que cuenta la herramienta son: Empresa, Contactos, Proyecto, Calendario, Tareas. Ficheros, Foros, Informes, Tickets y Administración.
    - Permite la asignación de recursos (equipamientos, mobiliario...) a un proyecto.
    - Permite clasificar y/u ordenar los proyectos en función de su estado: En curso, pendientes, cerrados.
    - Permite la visualización de informes y estadísticas sobre los proyectos registrados,

ejemplo: Las horas asignadas (por usuario o proyecto), estado de un proyecto, estadísticas, etcétera.

### 3.2.1.6. Gestión Documental

Detalles:

- ◆ Nombre Completo: **Wiki**
- ◆ Versión a utilizar: **MediaWiki 1.12.0**
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: Es de código abierto, bajo la licencia GPL. El primer WikiWikiWeb fue creado por Ward Cunningham, quien inventó y dio nombre al concepto **wiki**, y produjo la primera implementación de un servidor WikiWiki para el repositorio de patrones del Portland (Portland Pattern Repository) en 1995.
- ◆ Tipo de Licencia: Software Libre
- ◆ URL en Internet: <http://www.c2.com/cgi/wiki>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos MINTUR de la facultad 3 y Catastro de la facultad1.
- ◆ Formación y Recursos Humanos Disponibles.
  - Estudiantes y Profesores capacitados.
  - Documentación on-line: <http://es.wikipedia.org/wiki/Wiki>
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Intel Pentium 300 MHz o superior (o equivalente)
  - Principales Características:
    - Permite que se escriban artículos colectivamente (co-autoría) por medio de un lenguaje de wikitexto editado mediante un navegador Web. Una página wiki singular es llamada "página wiki", mientras que el conjunto de páginas (normalmente interconectadas mediante hipervínculos) es "el wiki". Es mucho más sencillo y fácil de usar que una base de datos.
    - Facilidad con que las páginas pueden ser creadas y actualizadas. En general no hace falta revisión para que los cambios sean aceptados.
    - A veces se requiere hacer login para obtener una cookie de "wiki-firma", para autofirmar las ediciones propias. Otros wikis más privados requieren autenticación de usuario.
    - Cada página contiene un gran número de vínculos a otras páginas.
    - Permite al menos una búsqueda por títulos, a veces incluso una búsqueda por texto

completo.

- Algunos de los más utilizados son:
  - ✓ UseModWiki: el más antiguo, escrito en Perl.
  - ✓ MediaWiki: utilizado en todos los proyectos de Wikimedia. Basado en PHP y MySQL.
  - ✓ PhpWiki: basado en UseMod. Escrito en PHP, puede utilizar distintas bases de datos.
  - ✓ TikiWiki: CMS completo, con un wiki muy desarrollado, usando PHP y MySQL.
  - ✓ Otros como: MoinMoin, OpenWiking, WikkaWiki, DokuWiki.

### 3.2.1.7. Seguimiento de Errores

Detalles:

- ◆ Nombre Completo: Trac
- ◆ Versión a utilizar: 0.10.4
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: Edgewall Software
- ◆ Tipo de Licencia: BSD (Software Libre)
- ◆ URL en Internet: <http://trac.edgewall.org/>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Registros y Notarías, Identidad
- ◆ Formación y Recursos Humanos Disponibles.
  - Cursos, Manuales, Documentación que garantice el aprendizaje.
  - Estudiantes y Profesores capacitados.
- ◆ Información General:
  - Procesador Pentium a 233 megahercios (MHz) o mayor velocidad (se recomienda 300 MHz).
  - Al menos 64 megabytes (MB) de RAM (se recomienda 128 MB).
  - Principales Características.
    - Se administra vía Web, tiene herramientas para la gestión de proyectos que se le puede agregar mediante plug-ins, ejemplo: diagrama de Gantt.
    - Emplea una Wiki para documentar cualquier aspecto del proyecto.
    - Contiene sistema para definir y visualizar el estado de los hitos de un proyecto.
    - Manejo de eventos (*Timeline*)
    - Permite la realización de búsquedas. Pudiendo localizar páginas del wiki, comentarios dentro de los conjuntos de cambios o tickets en los que aparece una palabra.
    - Se integra con un sistema de control de versiones (Subversion).
    - Asociado al sistema de control de versiones permite ver los cambios que se han

producido en el programa de una forma visual (estado actual del repositorio, los cambios que se han ido produciendo, comparar distintas versiones.

- Permite realizar la gestión de varios aspectos de la administración de un proyecto: Configuración, Usuarios, Permisos, Plug-ins.
- Permite realizar notificaciones de cambios realizados en el proyecto, vía email.
- Interfaz externa para proyectos

### 3.2.2. Herramientas Verticales

#### 3.2.2.1. Herramientas de Modelado

Detalles

- ◆ Nombre completo: **Visual Paradigm for UML (CE)**
- ◆ Versión a utilizar: **6.1**
- ◆ Compañía, empresa, grupo que la creo y brinda soporte: Visual Paradigm International Company
- ◆ Tipo de Licencia: **Freeware**
- ◆ URL en internet: <http://www.visual-paradigm.com>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la Universidad. Docencia facultades de la UCI.
- ◆ Formación y Recursos Humanos Disponibles.
  - Cursos, Manuales, Documentación que garantice el aprendizaje. Docencia facultades de la UCI.
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Intel Pentium 300 MHz o superior (o equivalente)
  - Principales Características:
    - Permite hacer diagramas UML, generación automática de código, sincronización entre modelos y código entre otras posibilidades.
    - Permite la participación online de los miembros de un proyecto.
    - Lista de funciones:
      - ✓ Apoyo UML versión 2.1
      - ✓ Soporta BPMN
      - ✓ Generación de código (modelo de código, el diagrama de código)
      - ✓ Generación de diagrama de flujo de datos

- ✓ Diseño centrado en casos de uso y enfocado al negocio
- ✓ Ingeniería inversa para Java, C + +, XML Schema, XML NET exe / dll, CORBA IDL, Python y XML Schema
- ✓ VP de modelado en colaboración con el Servidor de trabajo en equipo, CVS, Subversion y Perforce
- Integración: NetBeans y Eclipse

### 3.2.2.2. Máquina Virtual de Java

#### Detalles

- ◆ Nombre completo: **JDK (Java Development Kit)**
- ◆ Versión a utilizar: **1.6.0\_3**
- ◆ Compañía, empresa, grupo que la creo y brinda soporte: Sun Microsystems
- ◆ Tipo de Licencia: **Desde Noviembre de 2006 esta bajo GPL**
- ◆ URL en internet: <http://java.sun.com/>
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la Universidad.
- ◆ Formación y Recursos Humanos Disponibles.
  - Cursos, Manuales, Documentación que garantice el aprendizaje. Docencia en facultades de la UCI.
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Procesador Intel Pentium 166 MHz o mejor (o equivalente)
    - Mínimo 64 MB de RAM.
    - 98 MB de espacio disponible en el disco duro para la instalación.
    - Windows versiones (2000 SP3+, XP Home, XP Professional SP1+, Server 2003 Edition, Vista)
  - Principales Características:
    - Es utilizado solo para ejecutar programas en Java.
    - Lista de funciones:
      - ✓ Soporte para lenguajes dinámicos y de scripting.
      - ✓ Mejora de las librerías y del compilador en tiempo de ejecución.
      - ✓ Mejoras en JIT, diversas optimizaciones del estilo visual de Swing y soporte para Windows Vista.

- JDK trae varios utilitarios:
  - ✓ El compilador java que transforma los archivos fuente .java a archivos con instrucciones para la máquina virtual .class
  - ✓ El intérprete java que es la propia máquina virtual.
  - ✓ Utilitarios como el appletviewer que permite probar un applet dentro de una página HTML, sin necesidad de tener un "browser", o el javadoc, que genera documentación de un programa a partir de su código fuente.
- Ofrece
  - ✓ Un compilador Java, capaz de generar Byte-Code.
  - ✓ Un JVM ("Java Virtual Machine"), capaz de ejecutar Byte-Code.
  - ✓ Un conjunto de Clases base utilizadas para generar programas Java.
  - ✓ Otras utilerías para administrar código escrito en Java.

### 3.2.2.3. Entornos de desarrollo integrados (IDE)

Detalles:

- ◆ Nombre Completo: **Eclipse**
- ◆ Versión a utilizar: **3.3.1**.
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: Eclipse Foundation.
- ◆ Tipo de Licencia: **Liberado originalmente bajo la *Common Public License*, pero después fue re-licenciado bajo la *Eclipse Public License (EPL)*.**
- ◆ URL en Internet: [www.eclipse.org](http://www.eclipse.org)
- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la Universidad.
  - Formación y Recursos Humanos Disponibles.
  - Cursos, Manuales, Documentación que garantice el aprendizaje.
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Procesador Power PC G4 o mejor.
    - Sistema operativo Windows, Linux, Solaris, QNX o Mac OS/X.
    - Mínimo 512 MB de RAM.
    - 138 MB de espacio disponible en el disco duro para la instalación.
    - JDK o JRE versión 1.5 o posterior. Se recomienda al menos la versión 1.5.1.

- Principales Características:
  - Emplea módulos (plug-ins) para proporcionar toda su funcionalidad
  - Provee:
    - ✓ Soporte para Java, CVS y Subversion (SVN)
    - ✓ Editor de texto y resaltado de sintaxis
    - ✓ Compilación en tiempo real
    - ✓ Pruebas unitarias con JUnit
    - ✓ Integración con Ant
    - ✓ Asistentes (wizards): para creación de proyectos, clases, tests, etc.
    - ✓ Refactorización
  - Asimismo, a través de "plug-ins" libremente disponibles es posible añadir:
    - ✓ Control de versiones con Subversion.
    - ✓ Integración con Hibernate, Spring, JSF, etcétera.

#### 3.2.2.4. Lenguaje de Programación

Detalles:

- ◆ Nombre del lenguaje: **Java**
- ◆ Información general:

Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990. Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución.

- Principales Características:
  - Sintaxis es muy parecida a la de C o C++, con un modelo de objetos más simple.
  - Gestión de hilos y ejecución remota.
  - Es un lenguaje multiplataforma, solo necesita la maquina virtual.
  - Es un lenguaje seguro: La máquina virtual, al ejecutar el código java, realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras, como por ejemplo los punteros.
  - Lenguaje robusto pues fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones principalmente en la compilación, además en tiempo de ejecución.
  - Java puede ser usado para crear dos tipos de programas: aplicaciones independientes y

applets.

- Interpretado y compilado a la vez. Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Se generan ficheros de clases compilados, pero estas clases compiladas son en realidad interpretadas por la maquina virtual de java. Es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).
- Multihilado. Java permite la creación de procesos internos conocidos como hilos (threads) lo que permite la concurrencia de tareas y acceso a recursos.
- Java interactúa con servidores HTTP mediante el manejo de documentos dinámicos JSP, acceso a servlets, XML, SOAP y applets.
- Java permite la programación de aplicaciones distribuidas gracias a su implícita distribución de objetos. La comunicación es entre objetos remotos (RMI: Remote Method Invocation, CORBA).
- Mediante máquinas virtuales para equipos móviles, Java permite programar equipos portátiles como celulares, PDAs, tarjetas inteligentes (JavaCards) y chips.

### 3.2.2.5. Frameworks

Como se ha visto en el capítulo anterior los frameworks son uno de los niveles de abstracción arquitectónica y por consiguiente el análisis de los que se proponen para la solución se hacen imprescindibles. A partir del organigrama de la arquitectura y de los estilos arquitectónicos seleccionados para la solución, se proponen cuatro frameworks fundamentales distribuidos de la siguiente forma:

- ◆ Para la presentación: **MyFaces 1.1.5**
- ◆ Motor de web services: **Apache Axis2 1.3**
- ◆ Negocio: **Spring 2.0**
  - Para la seguridad: **Acegi 1.0.7** que es un módulo de Spring
- ◆ Acceso a Datos: **Hibernate 3.2.0**

No es objetivo de los autores reinventar la rueda, por lo que los frameworks que se propusieron son soluciones que han sido probadas en la Universidad en diversos proyectos productivos en los cuales se han obtenido magníficos resultados.



❖ **MyFaces**

*MyFaces* es un desarrollo de código abierto creado por la fundación Apache de la especificación *Java Server Faces (JSF)* que permite programar superficies web con mayor funcionalidad y con más orientación al usuario. Básicamente, se trata de un marco de desarrollo de aplicaciones web siguiendo el patrón MVC, pero a diferencia de otros desarrollos que persiguen el mismo objetivo, como *Struts*; *MyFaces* tiene una fuerte orientación hacia considerar los componentes de la aplicación, botones, cajas de texto. Como ciudadanos de primera categoría, todo en *MyFaces* se construye alrededor de ellos, haciendo de esta forma que el desarrollo de las aplicaciones web sea parecido al de un programa creado para escritorio. (Torrice, et al., 2007)

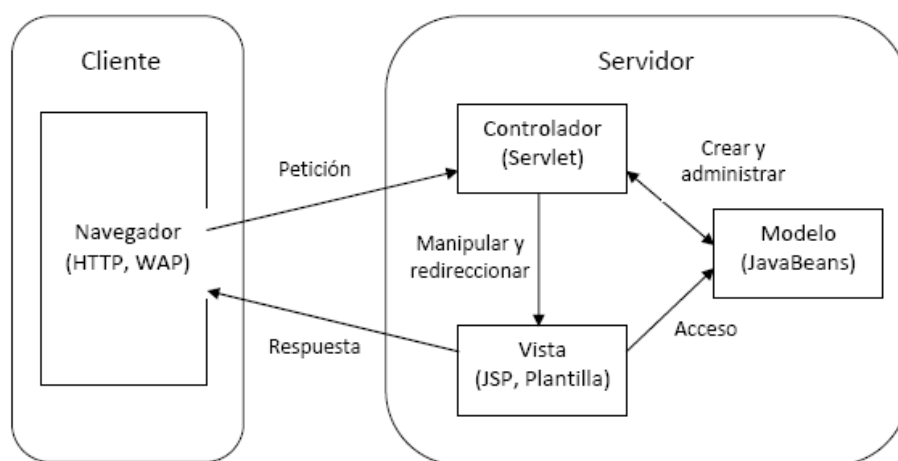


Figura 29. Arquitectura de una aplicación MyFaces (MyFaces, 2008)

Una característica que hace a *MyFaces* especialmente atractivo es el gran número de componentes adicionales que aporta, además de los exigidos por la especificación *JSF*, los programadores de *Jakarta* han hecho un magnífico trabajo creando un sin fin de componentes adicionales que son de gran utilidad en las aplicaciones web, menús, listas, etcétera. Estos componentes se agrupan en una librería que recibe en nombre de *tomahawk*. Estos componentes se podrían usar en cualquier distribución alternativa de *JSF* ya que están desarrollados siguiendo los estándares de construcción de componentes *JSF*.

*Tomahawk* es un componente que extiende de *MyFaces* y añade más funcionalidad que la descrita en la especificación de *JSF*. Tablas paginadas, árboles jerárquicos, ventanas pop-up, paneles, tablas maestro-detalle, calendario, agenda diaria, todos estos componentes forman parte de la librería *Tomahawk*. Por lo tanto, ¿para qué reinventar la rueda si ya se cuenta con componentes estándares que son ampliamente configurables y que facilitan el desarrollo de la capa front-end de las aplicaciones! (Torrice, et al., 2007).

Otro punto que hay que destacar de *MyFaces*, es que da la posibilidad de que estas validaciones se realicen tanto del lado del servidor como del cliente.

Realmente el uso de *MyFaces* no impide el uso de otras librerías de componentes, así que su uso depende tan solo de la utilidad que se encuentre en las funcionalidades que ofrece.

#### ❖ **Axis2**

Axis2 es un motor de Web Service desarrollado por la *Apache Software Foundation*, por lo tanto, es Open Source. Es una evolución de su predecesor, *Apache Axis*. Su diseño presenta una arquitectura modular que facilita la introducción de funciones adicionales y la integración de nuevas tecnologías de la pila WS.

#### **Módulos de Axis2:**

- ◆ Kandula, que proporciona WS-AtomicTransaction, WS-BusinessActivity y WS-Coordination.
- ◆ Sandesha, que implementa WS-ReliableMessaging
- ◆ Rampart, que incorpora WS-SecurityPolicy.
- ◆ Actualmente, se está desarrollando un nuevo módulo para soportar WS-Policy.
- ◆ Aunque la plataforma no permite trabajar directamente con UDDI, es posible integrar JUDDI con Axis2.

Tanto Axis2 como los módulos descritos, se distribuyen bajo licencia Apache 2.0.

#### **Axis2 proporciona:**

- ◆ La implementación del protocolo SOAP.
- ◆ Asistentes para trabajar con archivos WSDL.
- ◆ Deserialización del contenido XML (SOAP) en objetos Java que serán utilizados por el servicio y serialización de los objetos Java que devuelven el servicio en XML (SOAP). El programador sólo debe centrarse en la codificación de la lógica de negocio, y no debe preocuparse de la generación de los mensajes SOAP (Apache, 2007).
- ◆ Asistentes para la creación de esqueletos de servicio listos para desplegar de forma automática.
- ◆ Asistentes para la creación de objetos Java que modelan los mensajes SOAP.
- ◆ Mejora de rendimiento respecto a Axis1 usando la tecnología StAX (Streaming API for XML), fusiona las ventajas de SAX (rendimiento) y DOM (modelo 'pull parsing'), sin sus inconvenientes (modelo 'push parsing' de SAX y consumo excesivo de memoria de DOM) (Apache, 2007).

- ◆ Soporte del modelo de programación asíncrona y síncrona para consumir servicios web.
- ◆ Soporta diferentes protocolos de transporte (Chan, et al., 2007):
- ◆ HTTP, SMTP, TCP, JMS
  - Pudiendo indicar en cada sentido de la comunicación un transporte diferente.
- ◆ Combinando el modo de invocación con la flexibilidad de selección del protocolo de transporte en la petición y en la respuesta, se obtienen los siguientes cuatro patrones de invocación de servicios web (MEP: Message Exchange Pattern) (Apache, 2007):
  - Un canal de comunicación con bloqueo: llamada y bloqueo del cliente hasta que se responda. Puede esperarse o no respuesta del servidor.
  - Un canal de comunicación sin bloqueo: llamada y no bloqueo del cliente. Puede esperarse o no respuesta del servicio web. Es el nivel de asincronía más sencillo de implementar.
  - Dos canales de comunicación con bloqueo: llamada y bloqueo del cliente hasta recibir respuesta del servicio web por otro canal de comunicación.
  - Dos canales de comunicación sin bloqueo: llamada y no bloqueo del cliente. Cuando el servicio web responde se invoca a un método de retorno por otro canal de comunicación. Es el máximo nivel de asincronía y el más complejo de implementar.
- ◆ Soporta la creación de servicios web mediante Spring.
- ◆ Soporte completo para SOAP con adjuntos (SwA, SOAP with Attachments).
- ◆ El mapeo 'XML/SOAP' <--> 'Modelo Java del XML' se ha optimizado mediante AXIOM (AXIS Object Model) que usa StAX. Es el API de bajo nivel de Axis2 para manejo de mensajes SOAP (Chan, et al., 2007).
- ◆ El mapeo 'XML/SOAP' <--> 'Java Beans' se puede implementar mediante varias tecnologías: ADB (Axis2 Databinding Framework), XMLBeans, JAX-ME, JiXB y JaxBRI. De las cuales ADB es la más fácil de usar en Axis2y XMLBeans la que ofrece mayores prestaciones. Son los APIs de alto nivel para el manejo de servicios web. Anotar que JAX-ME y JaxBRI se soportan de manera "experimental". (Apache, 2007)

**Además esta plataforma integra:**

- ◆ SOAP 1.1/1.2 y REST
- ◆ WSDL 1.1/2.0
- ◆ WS-Reliable Messaging
- ◆ WS-Security
- ◆ WS-Coordination

- ◆ WS-AtomicTransaction
- ◆ MTOM, XOP, SOAP with attachments
- ◆ WS-Addressing
- ◆ WS-Policy
- ◆ SAAJ 1.1

### ❖ Spring

*Spring* es un framework de Java facilitará la creación de aplicaciones. Diseñado en módulos, con funcionalidades específicas y consistentes con otros módulos, facilita el desarrollo de funcionalidades específicas y hace que la curva de aprendizaje sea favorable para el desarrollador.

Spring facilita la manipulación de objetos, se usen EJBs o no, reduce la proliferación de *Singletons*, elimina la necesidad de usar distintos y variados tipos de ficheros de configuración, mejora la práctica de programación, permite el uso o no de EJBs, realizando el mismo tipo de funciones sin ellos.

#### Características de Spring:

- ◆ La inicial motivación era facilitar el desarrollo de aplicaciones J2EE, promoviendo buenas prácticas de diseño y programación. En concreto se trata de manejar patrones de diseño como: *Factory*, *Abstract Factory*, *Builder*, *Decorator* y *Service Locator*; que son ampliamente reconocidos dentro de la industria del desarrollo de software.
- ◆ Es código abierto.
- ◆ Enfoque en el manejo de objetos de negocio, dentro de una arquitectura en capas.
- ◆ Una característica de Spring es que puede actuar como pegamento de integración entre diferentes APIs (*JDBC*, *JNDI*, entre otros) y frameworks (por ejemplo entre *Struts* e *iBatis*).
- ◆ Objeto Viejo de Java (*POJO: Plain Old Java Object*) revaloriza la simplicidad de las clases Java aportando manejo de transacciones de forma no intrusiva.
- ◆ Configuración basada en archivos XML.
- ◆ Buen manejo de la Seguridad: como un requerimiento no funcional implementado como un aspecto (*AOP: Aspect Oriented Programming – Programación Orientada a Aspectos*) a través del framework *Acegi*.
- ◆ Remoting: Invocación de Método Remoto (*RMI: Remote Method Invocation*) simplificado, acceso y publicación de servicios web.
- ◆ Provee un paquete de pruebas específico para componentes del framework e integrado con *JUnit*.

Todas las funcionalidades que ofrece este framework están organizadas en siete grandes módulos que ofrece Spring:

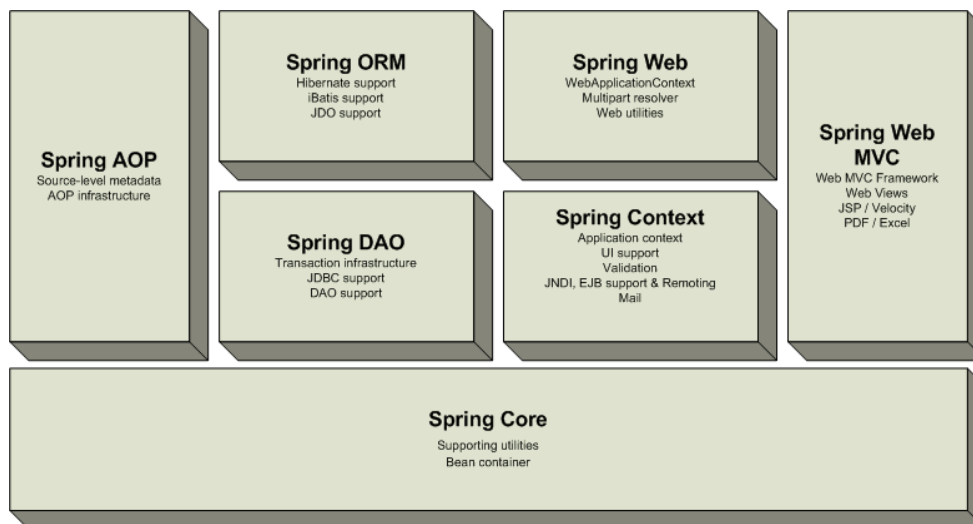


Figura 30. Diagrama de Módulos de Spring

1. El módulo **Core** o "Núcleo" es la parte fundamental del framework ya que provee toda la funcionalidad de Inyección de Dependencias permitiendo administrar la funcionalidad del contenedor de *beans*. El concepto básico de este módulo es el *BeanFactory*, que implementa el patrón de diseño *Factory* (fábrica) eliminando la necesidad de crear *singletons* programáticamente permitiéndote desligar la configuración y especificación de las dependencias de tu lógica de programación.
2. Encima del módulo **Core** se encuentra el módulo **Context** (Contexto), el cual te provee de herramientas para acceder a los beans de una manera elegante, similar a un registro JNDI. El paquete de contexto hereda sus características del paquete de *beans* y añade soporte para mensajería de texto, como son *resource bundles* (para internacionalización), propagación de eventos, carga de recursos y creación transparente de contextos por contenedores (como el contenedor de *servlets*, por ejemplo).
3. El paquete **DAO** provee una capa de abstracción de *JDBC* que elimina la necesidad de teclear código *JDBC* tedioso y redundante así como el parseo de códigos de error específicos de cada proveedor de base de datos. También, el paquete *JDBC* provee de una manera de administrar transacciones tanto declarativas como programáticas, no solo para clases que implementen interfaces especiales, pero para todos tus *POJOs* (por sus siglas en inglés, Viejos y simples objetos java).

4. El paquete **ORM** provee capas de integración para *APIs* de mapeo objeto - relacional, incluyendo, *JDO*, Hibernate e *iBatis*. Usando el paquete *ORM* se pueden usar esos mapeadores en conjunto con otras características que Spring ofrece, como la administración de transacciones mencionada con anterioridad.
5. El paquete **AOP** provee una implementación de programación orientada a aspectos compatible con AOP Alliance, permitiéndote definir *pointcuts* e interceptores de métodos para desacoplar el código de una manera limpia implementando funcionalidad que por lógica y claridad debería estar separada. Usando metadatos a nivel de código fuente se pueden incorporar diversos tipos de información y comportamiento al código, un poco similar a los atributos de .NET
6. El paquete **Web** provee características básicas de integración orientadas a la web, como funcionalidad multi-partes (para realizar la carga de archivos), inicialización de contextos mediante *Servlet listeners* y un contexto de aplicación orientado a web. Cuando se usa Spring junto con *WebWork* o *Struts*, este es el paquete que permite una integración sencilla.
7. El paquete **Web MVC** provee de una implementación *Modelo - Vista - Controlador* para las aplicaciones web. La implementación de Spring *MVC* permite una separación entre código de modelo de dominio y las formas web y permite el uso de otras características de Spring Framework como lo es la validación.

### Filosofía de Spring Framework

- ◆ Proveer un Framework no invasivo
- ◆ Siempre que se pueda reutilizar código
- ◆ Plug & Play de componentes
- ◆ Spring no desea reinventar la rueda

### Ventajas:

- ◆ Promueve de buenas prácticas de programación, ya que motiva a programar orientado a interfaces (que ya que empiezas no lo puedes dejar), realizar pruebas unitarias (es sencillísimo añadir objetos *Mock* con Spring), entre otras cosas.
- ◆ Disminuye la complejidad en el desarrollo, y sobre todo, simplifica y acelera las pruebas de unidad. No es necesario tener un Servidor de Aplicaciones para probar las clases que han sido desarrolladas a través de Spring, por lo que *JUnit* es suficiente para la ejecución de estas pruebas de unidad. Solo se necesita un IDE que pueda ejecutar estas pruebas (Eclipse, por ejemplo), el *XML* de configuración de Spring y los respectivos *JARs*; todo esto corriendo en una aplicación Java estándar.

- ◆ Permite que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos (Estos objetos pueden ser reutilizados tanto en entornos J2EE, aplicaciones standalone, entornos de pruebas,... sin ningún problema).
- ◆ La arquitectura en capas de Spring, ofrece cantidad de flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar la gestión de configuración basada en JavaBeans sin utilizar el framework MVC o el soporte AOP.

### Desventajas

- ◆ No es una especificación
- ◆ No es ligero.
- ◆ La configuración mediante XML es laboriosa.
- ◆ Es un framework complejo. Puede costar un poco de esfuerzo dominar cada una de las propiedades que caracterizan al mismo.

### ❖ Acegi

*Acegi Security System* es un framework de seguridad que ha sido diseñado fundamentalmente para que sea usado con Spring. Es Open Source, muy configurable, que permite reutilizar y portar componentes de seguridad. Básicamente se puede implementar fácilmente la seguridad y la autenticación de cualquier aplicación. Permite mantener la lógica de negocio libre de código de seguridad (seguridad no intrusiva), el mismo proporciona cuatro opciones principales de seguridad:

- ◆ Listas de control de acceso (ACL) web basadas en esquemas URL.
- ◆ Protección de métodos y clases Java usando AOP.
- ◆ Single sign-on (SSO)
- ◆ Seguridad proporcionada por el contenedor Web

La arquitectura de Acegi está fuertemente basada en interfaces y en patrones de diseño, proporcionando las implementaciones más comúnmente utilizadas y numerosos puntos de extensión donde nuevas funcionalidades pueden ser añadidas. Esta arquitectura puede hacer un poco difícil seguir el flujo de ejecución al principio, pero una vez comprendida la idea global se acepta como el precio necesario para poder disfrutar de un framework con una gran potencia. Como usa Spring para su configuración los usuarios de Spring tienen pocas dificultades con la configuración.

### ❖ Hibernate

Hibernate es un framework que constituye un motor de persistencia que implementa múltiples funcionalidades. El centro de la arquitectura de Hibernate lo constituyen una serie de interfaces que realizan el grueso de las funcionalidades del framework, dentro de ellas están:

Interfaces	Descripción
Session	Es la más usada en las aplicaciones, es la manejadora de la persistencia, a través de ella se pueden guardar y cargar objetos de la base de datos.
SessionFactory	Mediante ella se obtiene la Session.
Configuration	Es usada para configurar Hibernate se utiliza para especificar la ruta de los documentos de mapeo.
Transaction	Se utiliza para el control de las transacciones.
Query y Criteria	Permiten la ejecución de consultas a la Base de Datos.

Tabla 3. Interfaces de Hibernate

Además de estas interfaces Hibernate brinda otros aspectos muy importantes y que serán de gran utilidad en las aplicaciones, por ejemplo:

- ◆ La interface *Type*, es un elemento fundamental y muy poderoso, permite el mapeo de tipos javas a columnas en bases de datos incluso a varias columnas, incluye tipos como Calendar, *byte []*, y permite además definir tipo de datos propios (Persona, Dirección, Nombre).
- ◆ Las interfaces *Callback* gestionan el ciclo de vida de los objetos (*Lifecycle*, *Validatable*).
- ◆ Un dialecto orientado a objetos Lenguaje de Consultas de Hibernate (*HQL: Hibernate Query Language*) fácil de manejar y familiar al Lenguaje de consulta estructurado (*SQL: Structured Query Language*) que aunque no es un lenguaje de manipulación de datos como este, puesto que es usado solamente para extraer datos no para borrar, insertar o actualizar, permite realizar complejas consultas.

Hibernate puede ser configurado y ejecutado en casi cualquier aplicación java y entorno desarrollo. Generalmente es usado en aplicaciones cliente-servidor aunque también se puede utilizar en aplicaciones basadas en Swing y SWT. Define dos tipos de configuración:

- ◆ Entorno manejado, entornos que proveen reservas de recursos como conexiones a base de datos (*connections pool*), transacciones y seguridad declarativa, en este caso se encuentran los servidores de aplicación como *JBoss*, *BEA WebLogic* entre otros.
- ◆ Entorno no manejado, es el caso de los contenedores de *servlet* como *Tomcat*, las aplicaciones de escritorio también son incluidas aquí; este tipo de entorno no provee transacciones automáticas ni manejos de recursos, la propia aplicación realiza el manejo de las conexiones a base de datos.



En los entornos no manejados Hibernate se ocupa de las transacciones y las conexiones *JDBC* o lo deja para que el desarrollador se ocupe de esto, en el caso de los entornos manejados Hibernate se integra con el contenedor para ocuparse de los *DataSources* y de las transacciones. Salvo estas diferencias lo demás es común para cualquier entorno.

Otros aspectos que son esenciales en el desarrollo de aplicaciones con Hibernate lo constituyen los ficheros de mapeo y configuración. Para cada clase persistente se le hace corresponder un fichero de mapeo que es el lugar donde se le especifica al framework como va a persistir dicha clase en la base de datos pero además se trata todo lo referente a las relaciones de esta clase con otras incluyendo el tratamiento de herencia y polimorfismo. Ahora el archivo de configuración es el que le dice a Hibernate todo lo referente a la conexión que se puede alcanzar vía *JNDI* en el caso de los entornos manejados o cargando la conexión con el driver correspondiente al gestor haciendo uso si se quiere de alguna herramienta que maneje la reserva de conexiones todo esto en el mismo archivo.

Estos ficheros son el alma del mapeo objeto relacional, son los que proveen toda la información del modelo de objetos que hace posible llevarla al modelo de las bases de datos.

Hibernate presenta un mecanismo persistente totalmente transparente, las clases desconocen su capacidad de persistir, no necesitan extender comportamientos especiales de otra clase o interface, no necesitan un contenedor de aplicaciones, pueden tener lógica de aplicación, son de fácil manejo y pueden ser usadas para transportar los datos a cualquier capa de la aplicación (patrón *ValueObject*), estas son algunas de las razones por las cuales este framework ha ganado tanto en popularidad y preferencia por encima de los EJBs de entidad de SUN y otras herramientas objeto/relacional.

#### **Ventajas:**

- ◆ Abstrae del manejo del código SQL, haciéndonos ganar en tiempo de desarrollo.
- ◆ No necesita ningún entorno especial para correr sea servidor aplicaciones, contenedores, etc.
- ◆ Las clases que persisten no tienen que implementar ninguna interfaz especial ni extender ninguna clase, por lo que son completamente transparentes haciendo posible su reutilización en distintas partes de la aplicación con el objetivo de transportar los datos, y haciéndola desacoplada.
- ◆ Permite manejar las transacciones de una manera eficaz.
- ◆ Toda la configuración de la conexión y el mapeo se realiza en ficheros externos, haciendo transparente el uso de distintos gestores
- ◆ Permite manejo de almacén de conexiones (*Connections Pool*).
- ◆ Es un framework open source y gratis.

### Desventajas

- ◆ No es un estándar J2EE.
- ◆ Es un framework complejo. Puede costar un poco de esfuerzo dominar cada una de las propiedades que caracterizan al mismo.

#### 7.1.1.1. Servidores de Aplicaciones

##### Servidor Web

Detalles:

- ◆ Nombre Completo: **Tomcat**
- ◆ Versión a utilizar: **6.0.13**
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: Apache Software Foundation.
- ◆ Tipo de Licencia: **Apache 2.0 Licence**
- ◆ URL en Internet: <http://tomcat.apache.org/>
  - Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la Universidad.
  - Formación y Recursos Humanos Disponibles: Cursos, Manuales, Documentación que garantice el aprendizaje.
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Mínimo 175 MB de RAM o mayor.
    - JDK o JRE versión 5.0 (1.5.x) o posterior. Al menos la versión 1.5.1
  - Principales Características:
    - Tomcat es un contenedor de servlets utilizado como la implementación de referencia oficial para las tecnologías de Java Server Pages y Java Servlet.
    - Implementado a partir de las especificaciones Servlet 2.5 y JavaServer Pages (JSP) 2.1
    - Recarga de servlets y funciones básicas HTTP
    - Implementado de Servlet 2.5 y JSP 2.1
    - Soporte para Unified Expression Language 2.1
    - Diseñado para funcionar en Java Platform, Standard Edition 5.0 y posteriores.
    - Soporte para Comet a través de la interfaz CometProcessor
    - Ventajas:
      - ✓ Tomcat es gratis, es fácil de instalar.

- ✓ Se ejecuta en máquinas más pequeñas y es compatible con las API más recientes de Java.
- ✓ Puede descargarse, instalarse y probarse. Es fiable.
- ✓ Tomcat ocupa muy poco espacio, teniendo su código binario (todas clases de Java), de modo que no es raro que se ejecute tan deprisa.
- ✓ Compatibilidad con J2EE. Tomcat no es directamente compatible con todas las API de J2EE (como JDBC, JNDI, JavaMail, RMI, JMS, XML y EJB), todas esas API, con la notable excepción de EJB, están disponibles agregando archivos Java (JAR) disponibles gratuitamente.

### Servidor de Aplicaciones

Detalles:

- ◆ Nombre completo: Apache (Servidor de aplicaciones J2EE)
- ◆ Versión a utilizar: **Apache 2.2.4**
- ◆ Compañía, Empresa, Grupo de la creó y brinda soporte: Apache Software Foundation.
- ◆ Tipo de licencia: **Licencia Apache**
- ◆ URL en internet: <http://httpd.apache.org/>
- ◆ Referencias de estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la Universidad.
- ◆ Formación y recursos humanos disponibles:
  - Cursos, manuales, documentación que garantice el aprendizaje.
- ◆ Información general:
  - Requerimientos de hardware
    - Mínimo 128 MB de RAM o superior.
    - Procesador Procesador Power PC G4 o mejor.
    - Sistema operativo Windows o Linux
    - 50 MB de espacio disponible en el disco duro para la instalación.
  - Principales Características:
    - De código abierto implementado en Java puro.
    - Mensajes de error altamente configurables.
    - Bases de datos de autenticación y negociado de contenido.
    - Apache tiene amplia aceptación en la red: en el 2005, Apache es el servidor HTTP más usado, siendo el servidor HTTP del 48% de los sitios web en el mundo y decreciendo su

cuota de mercado (estadísticas históricas y de uso diario proporcionadas por Netcraft).

- La arquitectura del servidor Apache es muy modular. Algunos de estos son:
  - ✓ mod\_ssl - Comunicaciones Seguras vía TLS.
  - ✓ mod\_rewrite - reescritura de direcciones servidas (generalmente utilizado para transformar páginas dinámicas como php en páginas estáticas html para así engañar a los navegantes o a los motores de búsqueda en cuanto a como fueron desarrolladas estas páginas).
  - ✓ mod\_dav - Soporte del protocolo WebDAV.
  - ✓ mod\_deflate - Compresión transparente con el algoritmo deflate del contenido enviado al cliente.
  - ✓ mod\_auth\_ldap - Permite autenticar usuarios contra un servidor LDAP.
  - ✓ mod\_proxy\_ajp - Conector para enlazar con el servidor Jakarta Tomcat de páginas dinámicas en Java (servlets y JSP).
- Ventajas:
  - Modular
  - Open source
  - Multi-plataforma
  - Extensible
  - Popular (fácil conseguir ayuda/soporte)
  - Gratuito

### 7.1.1.2. Sistema Gestor de Base de Datos

Detalles:

- ◆ Nombre Completo: **PostgreSQL**
- ◆ Versiones a utilizar:
  - Servidor **8.2.5**
  - Cliente Administración: **PGAdmin III 1.6.3**
- ◆ Compañía, Empresa, Grupo que la creó y brinda soporte: Equipo de desarrolladores (voluntarios en su mayoría) dispersos alrededor del mundo y comunicados vía Internet. Este es un proyecto de la comunidad y no es controlado por ninguna compañía.
- ◆ Tipo de Licencia: **BSD**
- ◆ URL en Internet: <http://www.postgresql.org/>

- ◆ Referencias de Estudios de Casos y/o Proyectos donde se ha utilizado: Proyectos de la Universidad. Docencia facultades de la UCI.
  - Formación y Recursos Humanos Disponibles: Cursos, Manuales, Documentación que garantice el aprendizaje. Documentación on-line.
- ◆ Información General:
  - Requerimientos Mínimos de Hardware:
    - Intel Pentium 300 MHz o superior (o equivalente)
  - Principales Características:
    - Soporta casi toda la sintaxis SQL.
    - Multiplataforma.
    - Integridad transaccional.
    - Acceso concurrente multiversión, MVCC Control de Concurrencia Multi-Versión.
    - Cliente/Servidor: Usa una arquitectura proceso-por-usuario cliente/servidor.
    - Write Ahead Logging incrementa la dependencia de la base de datos al registro de cambios antes de que estos sean escritos en la base de datos.
    - Soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL.
    - Herencia de tablas.
    - Incluye la mayoría de los tipos de datos SQL92 y SQL99 (INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, y TIMESTAMP).
    - Soporta almacenamiento de objetos grandes binarios, además de tipos de datos y operaciones geométricas.
    - Puntos de recuperación a un momento dado.
    - Sofisticado analizador/optimizador de consultas.
    - Soporta juegos de caracteres internacionales, codificación de caracteres multibyte.
    - Límites:
      - ✓ Máximo tamaño de base de datos ilimitado.
      - ✓ Máximo tamaño de tabla 32 TB.
      - ✓ Máximo tamaño de tupla 1.6 TB.
      - ✓ Máximo tamaño de campo 1 GB.
      - ✓ Máximo tuplas por tabla ilimitado.
      - ✓ Máximo columnas por tabla 250 - 1600 dependiendo de los tipos de columnas.
      - ✓ Máximo de índices por tabla ilimitado.

- Integración: NetBeans y Eclipse

## 7.2. Calidad de la Arquitectura

La determinación de la Arquitectura de Software consiste en la toma de decisiones respecto a: la organización del sistema de software, la selección de los elementos estructurales y sus interfaces, comportamiento de estos elementos estructurales, la posible composición de los elementos estructurales en subsistemas más grandes, el estilo que guía esta organización, entre otros, teniendo en cuenta las propiedades del sistema de software que se quieren lograr como: seguridad, portabilidad, escalabilidad, interoperabilidad. Por lo que se puede concluir que la Arquitectura de Software es una disciplina que se ocupa más por la capacidad del sistema que por la funcionalidad del mismo ya que ella responde a los requisitos no funcionales.

En el caso particular de esta propuesta de arquitectura los atributos de calidad que se propusieron resolver fueron los siguientes:

### ◆ Observables

- Seguridad: Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos. Por otro lado se puede ver también como la medida de ausencia de errores que generan pérdidas de información.

Este atributo de calidad es muy importante puesto que un usuario no puede confiar en los datos de un sistema que no le ayude a controlar el acceso de personas no autorizadas o a detectar errores de operación en los que se introducen y generan datos erróneos.

La seguridad del sistema se manejó de diferentes maneras:

- La comunicación de servicios web mantiene su integridad y confidencialidad con el estándar *ws-security* a través del modulo *Rampart* del framework *Axis2*.
- Los servicios ofrecidos por el sistema solo pueden ser accedidos por sistemas o servicios autorizados, de esta forma existe un sistema de autenticación para los servicios web transparente al usuario de la aplicación.
- La comunicación entre los componentes de la plataforma y el manejo de datos del sistema son cifrados mediante algoritmos de encriptación, específicamente, el manejo de la seguridad de claves de acceso de usuarios.
- La arquitectura de despliegue que se propuso también mantiene la seguridad, ya que para acceder al repositorio de servicios y a los servidores web se deberá atravesar un firewall de

software configurado de tal manera que por defecto deniegue todas las entradas, aceptando solo las direcciones que se especifiquen explícitamente y el puerto por el cual se podrán conectar. A la base de datos solo podrán acceder los servidores web, de modo que en el fichero de configuración del PostgreSQL se le define que servidores específicos interactuarán con él, que en este caso serán los servidores web. También para acceder al sistema, independientemente de la autenticación, solo harán aquellas máquinas clientes que se especifiquen explícitamente, pues para acceder a la misma habrá que atravesar un firewall de software configurado de la misma forma de denegar todo excepto lo que se le especifique explícitamente.

- Se utilizarán usuarios con roles bien definidos para acceder a la base de datos, para cada una de las operaciones del sistema.
- Para garantizar la seguridad de la Base de Datos, ningún usuario tendrá privilegios de administración.
- El módulo de Administración del sistema es el encargado de gestionar los usuarios que van a interactuar con el sistema, así como sus privilegios.
- El sistema contendrá disparadores (*Triggers*) en la Base de Datos para no permitir la manipulación de los datos directamente en el SGBD.
- Parte de la seguridad correrá por parte de la tecnología Java utilizada para el desarrollo de la aplicación.
- El sistema generará copias de seguridad, periódicas y automáticas, de los datos contenidos en el mismo.
- El sistema permitirá restaurar por parte de técnicos especializados la información del sistema a partir de una copia de seguridad anterior.
- El sistema gestionará la concurrencia de datos, de manera que al actualizar un registro, se comprueba si los datos que se desean modificar no han sido modificados previamente por otro sistema.
- En todo momento en el sistema se tendrá constancia de quién, desde dónde, y cuándo se realizó una operación determinada en el sistema.
- Parte de la seguridad vendrá dada en la realización de un fuerte manejo de excepciones.

#### ◆ No Observables

- Portabilidad: Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación como: hardware, software o una combinación de los dos. La portabilidad es mayor cuanto menor es su dependencia del software de plataforma.

El Sistema de Gestión de Inventarios al estar desarrollado en java y sobre la web es independiente de plataforma. De modo que sus únicas restricciones de software para poder ejecutarse en cualquier sistema operativo son: la máquina virtual de java y el servidor de web. Por otra parte las estaciones clientes que se deseen conectar con el sistema solo necesitan disponer de un navegador web.

- Escalabilidad: Es el grado con el que se puede ampliar el diseño arquitectónico, de datos o procedimental. En general, la escalabilidad hace referencia a la capacidad del sistema para mantener, si no mejorar, su rendimiento medio conforme aumenta el número de clientes. Se trata de un atributo del sistema que procede de la combinación de todos los demás atributos, la implementación y el diseño general, así como del modelo de interacción que se elija y no se trata de una propiedad del sistema que se pueda activar y desactivar mediante programación o que se pueda de alguna forma controlar directamente.

La escalabilidad constituye factor influyente en el crecimiento de un sistema. Un sistema que necesita crecer es un sistema con un rendimiento actual que no dispone del número de usuarios esperado. ¿Cómo se puede mejorar estructuralmente su rendimiento? Hay dos opciones:

- Con un hardware de mayor potencia (Escala en sentido ascendente): Significa básicamente que se realiza la migración de todo bajo la protección de un nuevo sistema de hardware mucho más eficaz. El impacto en el código existente y la organización del sistema es mínima (Mendoza, 2004).
- Con una mejor combinación de hardware y software (escala hacia afuera): Aumenta la potencia de procesamiento del sistema en su globalidad, es intrínsecamente modular y está formado por una agrupación de equipos. El total de la potencia de procesamiento es la suma de la velocidad física de cada equipo transferida por la partición de aplicaciones y datos extendida a través de los nodos (Mendoza, 2004).

La arquitectura propuesta tiene las ventajas que le brinda el estilo arquitectónico que utiliza, una Arquitectura Orientada a Servicios estructurada en capas. Al estar estructurado en capas el sistema, la capa de servicios y la de datos pueden desplegarse en diferentes nodos. Por otra parte, los nodos donde estén desplegadas estas capas pueden escalar ascendentemente mejorándole las prestaciones como: memoria RAM y microprocesador. También el sistema podrá escalar hacia afuera agregándole recursos de procesamiento en el nivel medio así como en el nivel de datos para adaptarse al creciente número de usuarios y cargas de trabajo.



La instalación de la aplicación esta prevista para un total de siete nodos en un inicio, y en dependencia de la demanda y el crecimiento de los usuarios, podrá escalar tanto de manera ascendente como hacia afuera. Donde la configuración seria de la siguiente forma: un primer nodo donde estará la capa de presentación de la aplicación web, un segundo en el que se encontrará el repositorio de los servicios brindados (*UDDI*), en el nodo tres se encontrará un servidor de aplicaciones Apache cuya función es la de balancear la carga entre los servidores web (*Apache Tomcat*) cuatro y cinco para manejar la concurrencia y el rendimiento, estos últimos van a contener los servicios proporcionados por el sistema, y finalmente en el nodo seis se encontrará Sistema Gestor de Base de Datos el cual almacenará toda la información generada por el sistema e implementará un mecanismo de réplica de su información con el séptimo nodo el cual contendrá un servidor de respaldo previendo posibles fallas de la base de datos.

- Interoperabilidad: Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de *integrabilidad*.

En pocas palabras, la interoperabilidad es una vía ya probada para resolver la diversidad y heterogeneidad que ofrece el mercado. Internet es, probablemente, el ejemplo más obvio de este tipo de interoperabilidad, donde cada pieza de software puede conectarse e intercambiar datos en la medida en que respete los protocolos fundamentales.

La interoperabilidad del sistema viene dada por parte de los servicios web que son desarrollados basados en los estándares de la web como: XML, SOAP, WSDL y UDDI. Lo cual posibilita que cualquier otra aplicación pueda interactuar de forma correcta con el Sistema de Gestión de Inventario con un conocimiento, mínimo o nulo, unos de otros, sin importar cómo ni en que entorno fueron desarrollados.

- Integrable: Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.

La integrabilidad depende de:

- Complejidad de las componentes
- Mecanismos y protocolos de comunicación
- Claridad en la asignación de responsabilidades
- Calidad y completitud de la especificación de las interfaces

El Sistema de Gestión de Inventario asegura su integración con otras aplicaciones ya que expone toda su lógica de negocios mediante servicios web que son publicados en el repositorio de servicios, de modo que cualquier otro servicio o aplicación que los necesite pueda usarlos. De esta forma se puede interactuar con otros sistemas de software o extender el sistema con servicios adicionales.

La calidad no tiene sentido sin evaluación, pero en la práctica, en pocos casos se evalúa debido a que evaluar es difícil por: existir atributos de calidad que aún no son bien entendidos, muchas arquitecturas tienen alto nivel de abstracción, descripciones arquitecturales informales y falta de experiencia en cuestiones de evaluación (Astudillo, 2004).

En la Universidad de las Ciencias Informáticas actualmente no se cuenta con un mecanismo para la evaluación de Arquitecturas de Software en los proyectos productivos. Esto viene dado, en gran parte, por la falta de experiencia en estas cuestiones, sobre todo en la práctica, y por la novicia de la misma, pues solo cuenta con seis años de creada. Debido a esto, y por cuestiones de tiempo y espacio, no es objetivo de los autores realizar una evaluación de la arquitectura propuesta en este documento. Aunque sí se recomienda por parte de los mismos, evaluarla mediante alguno de los métodos de evaluación de Arquitecturas de Software propuestos por el *Software Engineering Institute (SEI)*, como: el *ATAM*.

### 7.3. Ventajas y desventajas de la arquitectura propuesta

La arquitectura propuesta como todo tipo de arquitectura tiene sus ventajas y desventajas. Es objetivo de los autores realizar un análisis de estas, puesto a que contribuirán con la formación del basamento necesario para realizar su valoración final de la misma.

#### Ventajas

Varias son las ventajas que se derivan de la arquitectura propuesta y gran parte de ellas se deben a la selección del estilo SOA y su estructuración en capas, ya que se heredan todas las características y ventajas que brindan estos estilos:

- ◆ Presenta las ventajas que brindan los servicios web, entre ellas están:
  - Al apoyarse en HTTP, los servicios web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.
  - Los servicios web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
- Bajo acoplamiento puesto a que los clientes no necesitan conocer nada acerca de la implementación de los servicios a los que están accediendo, salvo la definición WSDL.
- Independencia del lenguaje de programación puesto a que el servidor y el cliente no necesitan estar escritos en el mismo lenguaje
- ◆ Permite compartir y reutilizar componentes y servicios desarrollados.
- ◆ Permite al sistema evolucionar con la adición de nuevos servicios y su mejoramiento. Donde cada servicio evoluciona de una manera independiente.
- ◆ La flexibilidad que proporciona el acoplamiento débil entre los servicios. Cualquier cambio en la implementación de uno de ellos no afectaría al resto siempre que se mantenga la interfaz.
- ◆ La escalabilidad, puesto a que los servicios al estar débilmente acoplados existe muy poca dependencia entre las aplicaciones clientes y los servicios que usan.
- ◆ Al estar estructurado en capas el sistema cada una de sus partes es intercambiable de forma sencilla por otros componentes de software, siendo esto posible por el diseño arquitectónico que reduce las dependencias entre las capas.
- ◆ La estructuración en capas posibilita un mejor manejo de los errores ya que es fácil identificar en capa se encuentra el mismo.

Por otra parte la arquitectura propuesta al estar desarrollada sobre la plataforma J2EE cuenta con las ventajas que esta le brinda como:

- ◆ La modularidad del diseño arquitectónico:
  - **MyFaces** para los componentes de la capa de presentación.
  - **Axis2** como motor de servicios web.
  - **Spring** para la capa de negocio. Este framework me brinda también la facilidad de integrarse con Acegi para la seguridad e Hibernate para la capa de acceso a datos.
  - **Acegi** para la seguridad de los servicios desarrollados con Spring y posteriormente exportados como servicios web mediante Axis2.
  - **Hibernate** para la capa de acceso a datos.

Esta modularidad viene dada también por la estructura en capas en que está organizada la SOA y los servicios web.

- ◆ El empleo de frameworks reduce la cantidad de código ya que como se ha expuesto en capítulos anteriores los marcos de trabajo se pueden considerar aplicaciones genéricas incompletas y configurables para desarrollar una aplicación concreta. Por lo que se puede decir que los frameworks permiten no tener que re-implementar componentes pre-elaborados, sino usarlos para conformar otros acorde a la finalidad que se necesite.
- ◆ Diseño arquitectónico concebido y desarrollado en java lo cual posibilita la portabilidad del producto de software sobre múltiples plataformas dependiendo sólo de la Máquina Virtual de Java, y en el caso de las máquinas clientes sólo se necesita un navegador web sin importar el Sistema Operativo.
- ◆ Las herramientas y frameworks propuestos para desarrollar la arquitectura son libres y se integran fácilmente mediante ficheros XML.
- ◆ El uso de ficheros de configuración proporcionan a la arquitectura un fundamento estable y portable ya que el código no necesita ser cambiado ante nuevas configuraciones de hardware, por ejemplo de base de datos.

### Desventajas

La arquitectura propuesta tiene varias desventajas como todo tipo de arquitecturas y no por ser menos significativas que las ventajas se dejarán de mencionar. Entre ellas se pueden ver:

- ◆ Al emplear el estilo arquitectónico SOA que se basa prácticamente en exponer toda la lógica de negocio mediante servicios web, adquieren de estos tanto sus beneficios como sus desventajas, algunas de estas últimas son:
  - Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.
  - Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI, CORBA, o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto.
- ◆ La configuración del entorno es compleja.
- ◆ Java no es un lenguaje fácil, aunque aquellos que tienen conocimientos de C o de C++, ya no tendrán tanto problema para su aprendizaje.
- ◆ El empleo de frameworks independientemente que trae ventajas como: la reutilización y la modularidad, muchas veces resulta problemático para los desarrolladores menos experimentados, puesto a que cada uno tiene su estructura y características.

- ◆ Las herramientas utilizadas al integrarlas con los frameworks necesarios consumen gran cantidad de recursos de la estación de trabajo del desarrollador.

#### 7.4. Valoración de la solución por parte de los autores

Resulta de extrema importancia analizar los resultados a los que se llega una vez que se ha concebido una arquitectura como la propuesta. Para lograr esto se hace necesario realizar un estudio crítico y riguroso de los beneficios y privaciones de la arquitectura descrita, como el efectuado en el apartado anterior. Por otra parte conforme en el proceso de desarrollo del software un sistema evoluciona las arquitecturas también lo hacen, y es menester de los autores velar que esta primera versión de la propuesta arquitectónica lo haga.

La arquitectura propuesta para el Sistema de Gestión de Inventarios se basa en la elaboración del artefacto Documento Descripción de la Arquitectura propuesto por la Universidad de las Ciencias Informáticas a emplear en los proyectos productivos. La propuesta arquitectónica va a ir evolucionando conforme lo haga dicho documento que recoge los aspectos más significativos de la arquitectura, y a la vez constituye una fuente de conocimiento y por qué no, también de entendimiento, puesto que ayudará tanto a desarrolladores como a clientes a entender cómo es que se va desarrollar el sistema.

Un aspecto donde se debe enfatizar y seguir trabajando para posibles mejoras y refinamientos, es en el uso de patrones de arquitectura y diseño para dar solución a problemas que surjan durante el desarrollo del sistema. En la descripción arquitectónica no se destaca la utilización de patrones de diseño, ya que los mismos deberán surgir a partir de problemas en el diseño de clases de la aplicación y quedarían a un nivel muy bajo de la abstracción quedando fuera de las actividades del arquitecto, aunque este es responsable de que el diseño de clases cumpla con los aspectos del diseño arquitectónico.

Otro aspecto a destacar es la modularidad del diseño arquitectónico de la propuesta que permitió el uso de frameworks específicos que contienen funcionalidades bien definidas para un determinado dominio de la aplicación. El uso de estos, va a permitir facilitar el desarrollo del software ya que diseñadores y programadores podrán pasar más tiempo identificando requerimientos de software que tratando con detalles de bajo nivel para proveer un sistema de software funcional. A pesar de esto hay que reconocer que los marcos de trabajo en muchas ocasiones añaden código innecesario y el tiempo que se pasaba el equipo de desarrollo diseñando y programando ahora lo emplea en aprender a usar los frameworks.

La tecnología y las herramientas a utilizar para el desarrollo del sistema desde el punto de vista vertical son todas libres, sin embargo influyen grandemente en el aumento de los costes del desarrollo del proyecto al demandar gran cantidad de recursos de hardware, por ejemplo: como mínimo 768 MB de memoria RAM (recomendado 1 GB) y un microprocesador Pentium IV para los desarrolladores al integrar todos los frameworks necesarios para el desarrollo del sistema.

A partir del análisis realizado anteriormente, de las ventajas y desventajas identificadas y examinadas, se considera que la arquitectura propuesta: responde a los principales puntos a medir, satisface los requisitos de los clientes, es construible por parte de los desarrolladores, puede ser probada y es administrable por los arquitectos y el jefe de proyecto.

### **7.5. Valoración de Especialistas**

Para la validación se realizó un aval firmado por especialistas (graduados de informática, relacionados con el tema de Arquitectura de Software), donde se estableció una valoración por parte de los mismos de varios atributos de calidad a cumplir por la solución, y cada uno de ellos le ofreció una puntuación del 0 al 10 del cumplimiento del aseguramiento en la propuesta arquitectónica de los atributos de calidad propuestos. Ver Anexo 17.

Estos atributos de calidad son variables cualitativas ordinales, pues llevan implícito diferencias de magnitud o intensidad entre sus categorías, que les confiere cierto orden. Se propone el análisis del comportamiento de las mismas, donde se ha establecido un nivel para cada variable, la medición de estas puede realizarse de varias formas: bien, regular, mal, o cualquier otro tipo de magnitud para medirla. En este caso se ha establecido una categoría para el nivel de madurez de cada variable, categoría que oscila entre 0 – 10. En el anexo 18 se muestran los resultados grafados de los avales de los seis especialistas consultados.

Para un mejor análisis de los datos obtenidos, se hace un análisis estadístico, utilizando como referencia el cálculo de la media aritmética sobre el resultado de los datos obtenidos según la escala propuesta (0-10). A continuación se muestra una representación gráfica de la media de las variables, a partir de los resultados obtenidos de los avales de los especialistas:



Figura 31. Representación de la media aritmética de los atributos de calidad

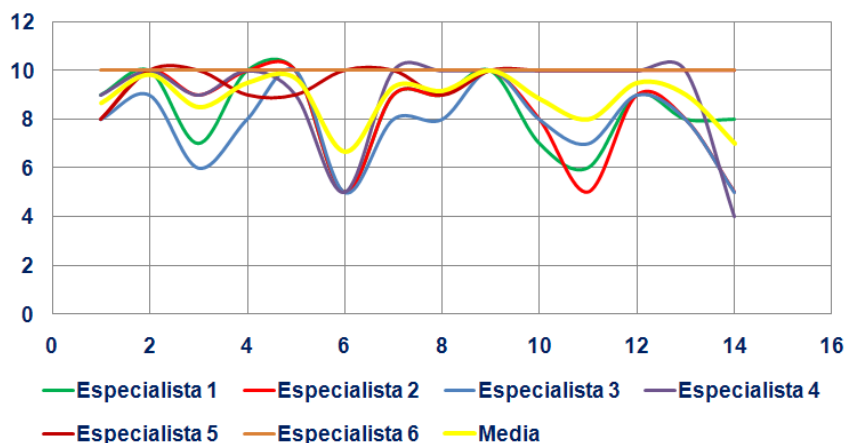


Figura 32. Representación de la media aritmética entre resultados individuales de los atributos

Según el análisis de los datos ofrecidos por los avales de los especialistas mencionados, se establece como regularidad positiva que las variables que se muestran a la derecha son las que mejor comportamiento tienen, obteniendo casi el máximo de aceptación según la escala propuesta. Estos resultados brindan una medida de cómo la arquitectura propuesta cumple con estos atributos de calidad.

Variable	Media	%
Modularidad del diseño	8.67	86.7
Bajo Acoplamiento	9.83	98.3
Normas y Estándares UCI	8.50	85
Escalabilidad	9.50	95
Portabilidad	9.67	96.7
Integrable	9.33	93.3
Modificabilidad	9.17	91.7
Evolución Constante	10.00	100
Alta Cohesión	8.83	88.3
Orientado a componentes	8.00	80
Reusable	9.50	95
Construable y Probable	9.00	90

Tabla 4. Promedio variables con regularidad positiva sobre solución.

Hay una regularidad a considerar, pues como mínimo se establecen en el 50% de los especialistas encuestados, donde el grado de aceptación propuesto por ellos sobre estas 2 variables a cumplir en la arquitectura es muy cercano a la media (5) de las escala de los datos (0-10), estableciéndose sobre las mismas una regularidad que impone mejorar en la descripción de la arquitectura la explicación de cómo es tratada la seguridad y del cubrimiento de los Conceptos de Arquitectura en la realización del trabajo.

Variable	1	2	3	4	5	6	Media	%
Seguridad	5	5	5	5	10	10	6.67	66.7
Cubrimiento de Conceptos de Arquitectura	8	5	5	4	10	10	7.0	70

Tabla 5. Promedios de Variables Seguridad y Cubrimiento de Conceptos de Arquitectura

A modo general las recomendaciones realizadas por los especialistas son aspectos a tener en cuenta para las recomendaciones del presente trabajo de diploma.

### 7.5.1. Premios otorgados a la solución

Como validación también se toma en consideración el premio de Relevante que se obtuvo con el presente trabajo en la VI Jornada Científica Estudiantil a nivel de Universidad. Para más detalle ver anexo 19.

## 7.6. Conclusiones Parciales

En este capítulo se complementó el artefacto Documento Descripción de la Arquitectura con la identificación de las herramientas que brindarán soporte al desarrollo del sistema, los frameworks a emplear y un análisis exhaustivo realizado a la propuesta de solución. Este análisis permitió corroborar que la Arquitectura de Software propuesta para el Sistema de Gestión de Inventarios satisface las restricciones arquitectónicas impuestas por los clientes y asegura el cumplimiento de los atributos de calidad: seguridad, portabilidad, escalabilidad e integrabilidad. La arquitectura propuesta presenta desventajas como todo lo desarrollado por la mano del hombre, pero que son mínimas en comparación con las ventajas que ofrece, y que con el refinamiento constante de la arquitectura poco a poco se irán contrarrestando.



## CONCLUSIONES

Como resultado de la investigación realizada en el presente trabajo de diploma se arribaron a las siguientes conclusiones:

- ◆ Se realizó un estudio profundo del estado del arte de la Arquitectura del Software que permitió conocer los enfoques actuales de la arquitectura, cómo evolucionaron, principales tendencias y definiciones, así como estilos arquitectónicos más significativos.
- ◆ Se definieron las principales tecnologías, frameworks, estilos y herramientas a utilizar en el desarrollo del sistema.
- ◆ Se realizó un estudio detallado de las principales categorías de patrones arquitectónicos, se fundamentó la elección de los mismos para su aplicación en el sistema.
- ◆ Se definió y describió la propuesta de arquitectura para el Sistema de Gestión de Inventario, siendo la misma construible por parte de los desarrolladores.
- ◆ Se realizó una validación a la solución mediante el criterio de especialista donde se identificaron las potencialidades de la solución y las debilidades según la percepción de los especialistas consultados.

## RECOMENDACIONES

Después de haber analizado las conclusiones a las que se llegan con la realización de este trabajo de diploma, se recomienda:

- ◆ Evaluar la arquitectura propuesta a partir de métodos de evaluación de calidad para ver cómo se comporta ante distintos escenarios que respondan a los atributos de calidad: seguridad, portabilidad, escalabilidad e integrabilidad; y de esta forma identificar sus debilidades.
- ◆ Refinar constantemente la arquitectura propuesta durante el ciclo de desarrollo del software, y para lo cual se propone:
  - Profundizar en el estudio de los servicios web y las Arquitecturas Orientadas a Servicios para de este modo hacer evolucionar la propuesta en versiones futuras incorporándole elementos tan significativos como el ESB y de esta forma ir escalando en los niveles de madurez de este tipo de arquitectura.
- ◆ Realizar un estudio de factibilidad y una evaluación de los costes de la tecnología a utilizar, por los recursos de hardware que demanda.
- ◆ Contemplar en próximas versiones varios escenarios de despliegue del sistema para el país.



## REFERENCIAS BIBLIOGRÁFICAS

- Albin, Stephen. 2003.** *The Art of Software Architecture: Design methods and techniques.* Nueva York : Wiley, 2003.
- Alur, D., Crupi, J. and Malks, D. 2001.** *Core J2EE Patterns, Best Practices and Design Strategies.* . s.l. : Prentice Hall, 2001.
- Alvez, Pablo, Foti, Patricia and Scalone, Marco. 2006.** *Proyecto Batuta - Generador de Aplicaciones Orquestadoras.* s.l. : Facultad de Ingeniería - Universidad de la República, 2006. Versión 1.2.1.
- Apache. 2007.** Apache - jUDDI. [Online] Apache Software Foundation, diciembre 11, 2007. [Cited: abril 17, 2008.] <http://ws.apache.org/juddi/>.
- **2007.** Apache Software Foundation: Axis2. [Online] 2007. [Cited: mayo 18, 2008.] <http://ws.apache.org/axis2/>.
- Astudillo, Hernán. 2004.** portal.inf.utfsm.cl - Arquitectura de Software Recuperación y evaluación. [Online] 2004. [Cited: abril 19, 2008.] [http://www.inf.utfsm.cl/~hernan/cursos/INF326-2004s1/bitacora/sesion\\_16.pdf](http://www.inf.utfsm.cl/~hernan/cursos/INF326-2004s1/bitacora/sesion_16.pdf).
- Bachman, Jon. 2005.** Sonic - Movin SOA On Up. [Online] septiembre 19, 2005. [Cited: abril 5, 2008.] [http://www.sonicsoftware.com/solutions/learning\\_center/soa\\_insights/movin\\_soa\\_on\\_up/index.ssp](http://www.sonicsoftware.com/solutions/learning_center/soa_insights/movin_soa_on_up/index.ssp).
- Bachmann, F., et al. 2000.** *Technical concepts of component-based software engineering.* s.l. : Software Engineering Institute, Carnegie Mellon University, 2000. Volume II, 2nd edition.
- Bass, L, Klein, M and Bachmann, F. 2000.** SEI. Quality Attribute Design Primitives. *Carnegie Mellon University.* [Online] 2000. [Cited: febrero 22, 2008.] <http://www.sei.cmu.edu/publications/documents/00.reports/00tn017.html>.
- Bass, Len, Clements, Paul and Kazman, Rick. 1998.** *Software Architecture in Practice.* s.l. : Addison-Wesley, 1998.
- Bengtsson, P. 1999.** University of Karlskrona. *Design and Evaluation of Software Architecture.* [Online] 1999. [Cited: febrero 20, 2008.] <http://www.ipd.hk-r.se/pob/archive/thesis.pdf>.
- Brey, Gustavo Andrés, et al. 2005.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires : Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas, 2005.
- Brown, A. W. and C.Wallnau, K. 1998.** *The current state of CBSE.* s.l. : IEEE Software, 1998. 15(5):37-46.
- Buschmann, F., et al. 1996.** *Pattern Oriented Software Architecture: A System of Patterns.* Inglaterra : John Wiley & Sons, 1996. 0471958697.

- Camacho, Erika, Cardeso, Fabio and Nuñez, Gabriel. 2004.** *Arquitecturas de Software. Guía de Estudio.* 2004.
- Casallas, Rubby and Villalobos, Jorge A. 2005.** Acis.org. *El actual ingeniero de software.* [Online] septiembre 2005. [Cited: febrero 15, 2008.] <http://www.acis.org.co/index.php?id=547>. Edición N° 93.
- Chan, Kathy and Mak, Andrew. 2007.** IBM - Introduction to Web Services Tools in WTP. [Online] mayo 3, 2007. [Cited: marzo 20, 2008.] <http://live.eclipse.org/node/251>.
- Cubillos, Jaime Andrés, et al. 2004.** Composición Semántica de Servicios Web. [Online] noviembre 2004. [Cited: abril 8, 2008.] [www.cintel.org.co/media/temacentral\\_3\\_14.pdf](http://www.cintel.org.co/media/temacentral_3_14.pdf).
- Endrei, Mark, et al. 2004.** IBM - Patterns: Service Oriented Architecture and Web Service. [Online] abril 2004. [Cited: febrero 25, 2008.] <http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246303.html?Open>. SG24-6303-00.
- Fowler, Martin, et al. 2002.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. 0-321-12742-0.
- García, Alberto Otero. 2007.** everis - Arquitecturas Empresariales. Orientación a Servicios (SOA) y Gestión de Procesos de Negocio (BPM). [Online] febrero 2007. [Cited: febrero 26, 2008.] <http://www.everis.com>.
- Garlan, David and Shaw, Mary. 1994.** *An introduction to software architecture.* s.l. : CMU Software Engineering Institute Technical Report, 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.
- Gómez, Omar Salvador Gómez. 2007.** *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* México : Brainworx S.A, 2007. 1870-0888.
- Keen, Martin, et al. 2004.** *Patterns: Implementing an SOA Using an Enterprise Service Bus.* New York : ibm.com/Redbooks, 2004. SG24-6346-00.
- Kruchten, P. 1999.** *The Rational Unified Process.* s.l. : Addison Wesley Longman, Inc, 1999.
- Lago, Ramiro Bagüés. 2007.** Proactiva Calidad - Patrón MVC. [Online] abril 2007. [Cited: marzo 17, 2008.] <http://www.proactiva-calidad.com/java/patrones/mvc.html>.
- Martinez, Marcela. 2007.** sonicsoftware: Modelo de Madurez en Arquitectura Orientada a Servicios(SOA). *SONIC.* [Online] 2007. [Cited: mayo 11, 2008.] <http://www.sonicsoftware.com>.
- Mateu, Carles. 2003.** Scribd - Desarrollo de Aplicaciones Web. [Online] diciembre 30, 2003. [Cited: abril 8, 2008.] <http://www.scribd.com/doc/2020380/Desarrollo-de-Aplicaciones-Web>.
- Mendoza, Victor Hugo Lamadrid. 2004.** infomatizate - Escalabilidad, un factor a tener en cuenta. [Online] abril 19, 2004. [Cited: abril 19, 2008.] [http://www.informatizate.net/articulos/escalabilidad\\_un\\_factor\\_a\\_tener\\_en\\_cuenta\\_19042004.html](http://www.informatizate.net/articulos/escalabilidad_un_factor_a_tener_en_cuenta_19042004.html).
- Microsystem, Sun. 2006.** Java en Castellano. [Online] 2006. [Cited: marzo 17, 2008.] <http://www.programacion.net/java/tutorial/patrones2/0/>.

- MSDN. 2004.** MSDN. *Layered application. Microsoft Patterns & Practices*. [Online] 2004. [Cited: febrero 15, 2008.] <http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpatterns/html/ArcLayeredApplication.asp>. Version 1.0.1.
- . **2006.** MSDN - Diseño de aplicaciones y servicios. *Patterns & Practices*. [Online] Microsoft Corporation, junio 26, 2006. [Cited: abril 28, 2008.] <http://www.microsoft.com/spanish/msdn/arquitectura/das/guias/AppArchCh1.msp#top>.
- MyFaces. 2008.** MyFaces Apache. [Online] abril 16, 2008. [Cited: mayo 11, 2008.] <http://myfaces.apache.org/index.html>.
- Natis, Yefim V. 2003.** Gartner. *Service-Oriented Architecture Scenario*. [Online] abril 16, 2003. [Cited: febrero 17, 2008.] AV-19-6751.
- OASIS. 2004.** OASIS - Introduction to UDDI: Important Features and Functional Concepts. [Online] octubre 2004. [Cited: abril 8, 2008.] <http://uddi.org/pubs/uddi-tech-wp.pdf>.
- Pamela Alejandra Martínez, Alejandra Argueta, Mayra Zavala, Federico Rodríguez, Victor Ordóñez.** *ERP Crecimiento Planificado de Sistemas de*. [Online] [Cited: 11 10, 2007.] [http://maestros.unitec.edu/~cariasas/ads/documents/ERP\\_VPERIISEM2001.pdf](http://maestros.unitec.edu/~cariasas/ads/documents/ERP_VPERIISEM2001.pdf).
- Perry, Dewayne E. and Wolf, Alexander L. 1992.** *Foundations for the study of software architecture*. s.l. : ACM SIGSOFT Software Engineering Notes, 1992. 17(4), pp. 40–52.
- Perry, Dewayne. 1997.** *Software Architecture and its relevance for Software Engineering*. s.l. : Coord'97, 1997.
- PostgreSQL. 2008.** PostgreSQL. [Online] 2008. [Cited: abril 25, 2008.] <http://www.postgresql.org/about/>.
- PostgreSQL. 2005.** postgresql - Chapter 12. Concurrency Control. [Online] 2005. [Cited: abril 24, 2008.] <http://www.postgresql.org/docs/8.1/static/mvcc.html>.
- Pressman, Roger S. 2002.** *Ingeniería de Software. Un enfoque practico*. s.l. : McGraw.Hill/Interamericana de España., 2002. 5.
- Reynoso, Carlos Billy and Kicillof, Nicolás. 2004.** MSDN Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. [Online] 2004. [Cited: 2 13, 2008.] [http://www.microsoft.co.ke/spanish/msdn/arquitectura/roadmap\\_arq/style.aspx](http://www.microsoft.co.ke/spanish/msdn/arquitectura/roadmap_arq/style.aspx).
- . **2004.** MSDN, Lenguajes de Descripción de Arquitectura (ADL). [Online] UNIVERSIDAD DE BUENOS AIRES, marzo 2004. [Cited: febrero 20, 2008.] [http://www.microsoft.com/spanish/msdn/arquitectura/roadmap\\_arq/lenguaje.msp#EKB](http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.msp#EKB).
- Reynoso, Carlos Billy. 2006.** MSDN Introducción a la Arquitectura de Software. [Online] 6 26, 2006. [Cited: 2 8, 2008.] [http://www.microsoft.com/spanish/msdn/arquitectura/roadmap\\_arq/intro.msp](http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.msp).

- . **2005**. MSDN webcast#1. *Seminario de Arquitectura de Software*. [Online] 2005. [Cited: febrero 18, 2008.] <http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx>.
- . **2005**. MSDN webcast#2. *Profundizando en Estilos de Arquitectura de Software*. [Online] 2005. [Cited: febrero 13, 2008.] <http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx>.
- . **2005**. MSDN webcast#3. *Arquitectura para distribución y agregación: Services Oriented Architecture (SOA)*. [Online] 2005. [Cited: febrero 16, 2008.] <http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx>.
- Sanchez, Alvaro Norberto Silva. 2006.** *Logística de Almacenamiento*. Caracas : Tecana American University, 2006. 2.
- Szyperski, Clemens. 2002.** *Component software: Beyond object-oriented programming*. s.l. : Addison-Wesley Pub Co, 2002. 2da edición.
- . **2000.** *Components and Architecture. Software Development*. 2000. online:10.
- Torrice, Gerber Fuentes and Cabrera, Lineth Cintya Camacho. 2007.** Postgrado en Informática: BenchMark sobre Tecnologías Faces. *pgi.umsa.bo*. [Online] septiembre 2007. [Cited: mayo 11, 2008.] [http://pgi.umsa.bo/enlaces/investigacion/pdf/INGSW3\\_50.pdf?PHPSESSID=ceef241421ecd67ac959046402fb7535](http://pgi.umsa.bo/enlaces/investigacion/pdf/INGSW3_50.pdf?PHPSESSID=ceef241421ecd67ac959046402fb7535).
- Velasco, Carlos Canal. 2000.** *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software*. Málaga : Universidad de Málaga, 2000.
- W3C. 2007.** W3C - SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation. [Online] abril 27, 2007. [Cited: abril 8, 2008.] <http://www.w3.org/TR/soap12-part1/#intro>.
- Welicki, León. 2007.** MSDN. *Patrones y Antipatrones: una Introducción - Parte II*. [Online] 2007. [Cited: febrero 23, 2008.] [http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ\\_3317.aspx](http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3317.aspx).

## BIBLIOGRAFÍA

- Agüero, Martín. 2007.** UP: Universidad de Palermo. Facultad de Ingeniería. *Introducción a Spring Framework*. [En línea] octubre de 2007. [Citado el: 17 de marzo de 2008.] [http://www.palermo.edu/ingenieria/downloads/introduccion\\_spring\\_framework\\_v1.0.pdf](http://www.palermo.edu/ingenieria/downloads/introduccion_spring_framework_v1.0.pdf). version 1.0.
- Albin, Stephen. 2003.** *The Art of Software Architecture: Design methods and techniques*. Nueva York : Wiley, 2003.
- Alur, D., Crupi, J. y Malks, D. 2001.** *Core J2EE Patterns, Best Practices and Design Strategies*. . s.l. : Prentice Hall, 2001.
- Alvez, Pablo, Foti, Patricia y Scalone, Marco. 2006.** *Proyecto Batuta - Generador de Aplicaciones Orquestadoras*. s.l. : Facultad de Ingeniería - Universidad de la República, 2006. Versión 1.2.1.
- Apache. 2007.** Apache - jUDDI. [En línea] Apache Software Foundation, 11 de diciembre de 2007. [Citado el: 17 de abril de 2008.] <http://ws.apache.org/juddi/>.
- . **2007.** Apache Software Foundation: Axis2. [En línea] 2007. [Citado el: 18 de mayo de 2008.] <http://ws.apache.org/axis2/>.
- Astudillo, Hernán. 2004.** portal.inf.utfsm.cl - Arquitectura de Software Recuperación y evaluación. [En línea] 2004. [Citado el: 19 de abril de 2008.] [http://www.inf.utfsm.cl/~hernan/cursos/INF326-2004s1/bitacora/sesion\\_16.pdf](http://www.inf.utfsm.cl/~hernan/cursos/INF326-2004s1/bitacora/sesion_16.pdf).
- Bachman, Jon. 2005.** Sonic - Movin SOA On Up. [En línea] 19 de septiembre de 2005. [Citado el: 5 de abril de 2008.] [http://www.sonicsoftware.com/solutions/learning\\_center/soa\\_insights/movin\\_soa\\_on\\_up/index.ssp](http://www.sonicsoftware.com/solutions/learning_center/soa_insights/movin_soa_on_up/index.ssp).
- Bachmann, F., y otros. 2000.** *Technical concepts of component-based software engineering*. s.l. : Software Engineering Institute, Carnegie Mellon University, 2000. Volume II, 2nd edition.
- Bagüés, Ramiro Lago. 2008.** Proactiva Calidad - Framework Spring. [En línea] enero de 2008. [Citado el: 17 de marzo de 2008.] <http://www.proactiva-calidad.com/java/spring/index.html>.
- Baragry, Jason y Reed, Karl. 2001.** *Why We Need a Different View of Software Architecture*. Amsterdam : The Working IEEE/IFIP Conference on Software Architecture (WICSA), 2001.
- Bass, L, Klein, M y Bachmann, F. 2000.** SEI. Quality Attribute Design Primitives. *Carnegie Mellon University*. [En línea] 2000. [Citado el: 22 de febrero de 2008.] <http://www.sei.cmu.edu/publications/documents/00.reports/00tn017.html>.
- Bass, Len, Clements, Paul y Kazman, Rick. 1998.** *Software Architecture in Practice*. s.l. : Addison-Wesley, 1998.
- Bengtsson, P. 1999.** University of Karlskrona. *Design and Evaluation of Software Architecture*. [En línea] 1999. [Citado el: 20 de febrero de 2008.] <http://www.ipd.hk-r.se/pob/archive/thesis.pdf>.



- Booch, G, Rumbaugh, J y & Jacobson, I. 1999.** *The UML Modeling Language User Guide*. s.l. : Addison-Wesley, 1999.
- Bosch, Jan. 2000.** *Design and use of Software Architecture*. s.l. : Addison-Wesley, 2000.
- Brey, Gustavo Andrés, y otros. 2005.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas*. Buenos Aires : Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas, 2005.
- Brown, A. W. y C.Wallnau, K. 1998.** *The current state of CBSE*. s.l. : IEEE Software, 1998. 15(5):37-46.
- Burbeck, Steve. 1992.** Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). *st-www.cs.uiuc.edu*. [En línea] 1992. [Citado el: 15 de febrero de 2008.] <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>.
- Buschmann, F., y otros. 1996.** *Pattern Oriented Software Architecture: A System of Patterns*. Inglaterra : John Wiley & Sons, 1996. 0471958697.
- Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. 2004.** *Arquitecturas de Software. Guía de Estudio*. 2004.
- Campo, Miguel Ángel Manzanedo del, y otros. 1999.** UBU - Guía de Iniciación al Lenguaje JAVA. [En línea] Universidad de Burgos , 19 de septiembre de 1999. [Citado el: 18 de marzo de 2008.] <http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/Index.htm>.
- Casallas, Rubby y Villalobos, Jorge A. 2005.** *Acis.org. El actual ingeniero de software*. [En línea] septiembre de 2005. [Citado el: 15 de febrero de 2008.] <http://www.acis.org.co/index.php?id=547>. Edición N° 93.
- Chan, Kathy y Mak, Andrew. 2007.** IBM - Introduction to Web Services Tools in WTP. [En línea] 3 de mayo de 2007. [Citado el: 20 de marzo de 2008.] <http://live.eclipse.org/node/251>.
- Chavarría, Raúl Eduardo. 2006.** slideshare: IDE ECLIPSE. *Ingeniería de Sistemas - Universidad de Antioquia*. [En línea] 2006. [Citado el: 1 de abril de 2008.] <http://www.slideshare.net/Benedeti/ide-eclipse-breve-gua-201399/>.
- Clement, Paul. 2000.** SEI (Software Engineering Institute). *SEI (Software Engeniering Institute)*. [En línea] agosto de 2000. [Citado el: 20 de noviembre de 2007.] <http://www.sei.cmu.edu/publications/documents/00.reports/00tn009.html>. Std. Z39-18298-102.
- Clements, Paul y Northrop, Linda. 1996.** *Software architecture: An executive overview*. s.l. : Technical Report, 1996. CMU/SEI-96-TR-003, ESC-TR-96-003.
- Cubillos, Jaime Andrés, y otros. 2004.** Composición Semántica de Servicios Web. [En línea] noviembre de 2004. [Citado el: 8 de abril de 2008.] [www.cintel.org.co/media/temacentral\\_3\\_14.pdf](http://www.cintel.org.co/media/temacentral_3_14.pdf).

- D, M. Shaw y Garlan. 1996.** *Introduction to Software Architectures. New perspectives on an emerging discipline.* Prentice Hall : s.n., 1996.
- Endrei, Mark, y otros. 2004.** IBM - Patterns: Service Oriented Architecture and Web Service. [En línea] abril de 2004. [Citado el: 25 de febrero de 2008.] <http://publib.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246303.html?Open>. SG24-6303-00.
- Evaluando Arquitecturas de Software. Parte 2. Métodos de Evaluación. Gómez, Omar Salvador Gómez. 2007.** 02, México : Brainworx S.A, 2007. 1870-0888.
- Fielding, Roy Thomas y Taylor, Richard. 2002.** *Principled design of the modern Web architecture.* s.l. : ACM Transactions on Internet Technologies, 2002. 2(2), pp. 115-150.
- Fowler, Amy. 2003.** Sun Microsystems. [En línea] 12 de abril de 2003. [Citado el: 17 de marzo de 2008.] <http://java.sun.com/products/jfc/tsc/articles/architecture/index.html>.
- Fowler, Martin, y otros. 2002.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. 0-321-12742-0.
- Froufe, Agustín. 1997.** Universidad de las Palmas de gran Canarias - Tutorial de Java. [En línea] 1 de enero de 1997. [Citado el: 26 de marzo de 2008.] <http://www.ulpgc.es/otros/tutoriales/java/Intro/tabla.html>.
- García, Alberto Otero. 2007.** everis - Arquitecturas Empresariales. Orientación a Servicios (SOA) y Gestión de Procesos de Negocio (BPM). [En línea] febrero de 2007. [Citado el: 26 de febrero de 2008.] <http://www.everis.com>.
- García, Ignacio, y otros. Enero 2005.** *Servicios Web.* Universidad de Castilla, La Mancha, España : s.n., Enero 2005. TIC2003-02737-C02-02.
- García, Sergio Campos. 2006.** - Hibernate. [En línea] 2006. [Citado el: 21 de marzo de 2008.] [http://md2.dei.inf.uc3m.es:8000/PA/Practicas/Exposiciones%20\(2006\)/Hibernate.ppt](http://md2.dei.inf.uc3m.es:8000/PA/Practicas/Exposiciones%20(2006)/Hibernate.ppt).
- Garlan, D. y Shaw, M. 1996.** *Introduction to Software Architectures. New perspectives on an emerging discipline.* s.l. : Prentice Hall., 1996.
- Garlan, David y Shaw, Mary. 1994.** *An introduction to software architecture.* s.l. : CMU Software Engineering Institute Technical Report, 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.
- Gómez, Omar Salvador Gómez. 2007.** *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* México : Brainworx S.A, 2007. 1870-0888.
- Gómez, Santiago Pavón. 2007.** dit UPM. *Universidad Politécnica de Madrid. Dpto. Ingeniería Sistemas Telemáticos.* [En línea] 2007. [Citado el: 18 de marzo de 2008.] <http://jungla.dit.upm.es/~santiago/docencia/apuntes/Swing/index.htm>. 060.423.
- Hanson, Jeff. 2006.** Programacion en Castellano. [En línea] 2006. [Citado el: 21 de marzo de 2008.] [http://www.programacion.net/articulo/jap\\_persis\\_hib/](http://www.programacion.net/articulo/jap_persis_hib/).

- Hernando, Diego Arranz. 2006.** Programación en castellano - Apuntes de XML. [En línea] 2006. [Citado el: 24 de marzo de 2008.] <http://www.programacion.net/tutoriales/XML/Apuntes de XML>.
- Jorge Triñanes, María de las Nieves Freira. 2006.** Gestión de Software. *Gestión de Software*. [En línea] 2006. [Citado el: 20 de octubre de 2007.] <http://www.fing.edu.uy/inco/cursos/gestsoft/>.
- Keen, Martin, y otros. 2004.** *Patterns: Implementing an SOA Using an Enterprise Service Bus*. New York : ibm.com/Redbooks, 2004. SG24-6346-00.
- klein, Mark. 2007.** SEI (Software Engineering Institute). *SEI (Software Engineering Institute)*. [En línea] Carnegie Mellon University, 2007. [Citado el: 18 de noviembre de 2007.] [http://www.sei.cmu.edu/architecture/ata\\_method.html](http://www.sei.cmu.edu/architecture/ata_method.html).
- Kruchten, P. 1999.** *The Rational Unified Process*. s.l. : Addison Wesley Longman, Inc, 1999.
- Lago, Ramiro Bagüés. 2007.** Proactiva Calidad - Patrón DAO. [En línea] abril de 2007. [Citado el: 17 de marzo de 2008.] <http://www.proactiva-calidad.com/java/patrones/DAO.html>.
- . 2007. Proactiva Calidad - Patrón MVC. [En línea] abril de 2007. [Citado el: 17 de marzo de 2008.] <http://www.proactiva-calidad.com/java/patrones/mvc.html>.
- Marañón, Gonzalo Álvarez. 1999.** Departamento de Tratamiento de la Información y Codificación - Qué es Java? [En línea] 1999. [Citado el: 26 de marzo de 2008.] <http://www.iec.csic.es/CRIPTONOMICON/java/funcionamiento.html>.
- Marcelo, G. Aruquipa Chambi y Edwin, P. Márquez Granado. 2007.** *Desarrollo de Software Basado en Componentes. Postgrado en Informática–Maestría en Ingeniería del Software*. La Paz; Bolivia : Universidad Mayor de San Andrés, 2007.
- Marset, Rafael Navarro. 2006-2007.** ELP-DSIC-UPV. *REST vs Web Services*. [En línea] 2006-2007. [Citado el: 16 de febrero de 2008.] <http://www.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>.
- Martinez, Federico y Abian, ablo. 2007.** harriegue - Hibernate. *Empresa especializada en Consultoría Informática. Innovacion Tecnologica*. [En línea] 2007. [Citado el: 21 de marzo de 2008.] <http://www.harriague.com.ar/hya/Portals/0/PresentacionUTN.pdf>.
- Martinez, Marcela. 2007.** sonicsoftware: Modelo de Madurez en Arquitectura Orientada a Servicios(SOA). *SONIC*. [En línea] 2007. [Citado el: 11 de mayo de 2008.] <http://www.sonicsoftware.com>.
- Mateu, Carles. 2003.** Scribd - Desarrollo de Aplicaciones Web. [En línea] 30 de diciembre de 2003. [Citado el: 8 de abril de 2008.] <http://www.scribd.com/doc/2020380/Desarrollo-de-Aplicaciones-Web>.
- Mendoza, Victor Hugo Lamadrid. 2004.** infomatizate - Escalabilidad, un factor a tener en cuenta. [En línea] 19 de abril de 2004. [Citado el: 19 de abril de 2008.] [http://www.informatizate.net/articulos/escalabilidad\\_un\\_factor\\_a\\_tener\\_en\\_cuenta\\_19042004.html](http://www.informatizate.net/articulos/escalabilidad_un_factor_a_tener_en_cuenta_19042004.html).

*Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC).*

**Aleksander González, Marizé Mijares, Luis E. Mendoza, Anna Grimán, María Pérez. 2005.** Colimia, Mexico : s.n., 2005, Vol. vol 1.

**Microsystem, Sun. 2006.** Java en Castellano. [En línea] 2006. [Citado el: 17 de marzo de 2008.] <http://www.programacion.net/java/tutorial/patrones2/0/>.

**Mitchell, Scott. 2003.** 4guysfromrolla - An Extensive Examination of Web Services: Part 1 . [En línea] 8 de octubre de 2003. [Citado el: 8 de abril de 2008.] <http://aspnet.4guysfromrolla.com/articles/100803-1.aspx>.

**MSDN. 2004.** MSDN. *Layered application. Microsoft Patterns & Practices.* [En línea] 2004. [Citado el: 15 de febrero de 2008.] <http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpatterns/html/ArcLayeredApplication.asp>. Version 1.0.1.

—. **2006.** MSDN - Diseño de aplicaciones y servicios. *Patterns & Practices* . [En línea] Microsoft Corporation, 26 de junio de 2006. [Citado el: 28 de abril de 2008.] <http://www.microsoft.com/spanish/msdn/arquitectura/das/guias/AppArchCh1.msp#top>.

**MyFaces. 2008.** MyFaces Apache. [En línea] 16 de abril de 2008. [Citado el: 11 de mayo de 2008.] <http://myfaces.apache.org/index.html>.

**Naranjo, Mauricio. 2007.** *Lineamientos para la adopcion de Arquitecturas Orientadas a Servicios para las empresas.* Bogotá D.C : Lucasian Labs, 2007.

**Natis, Yefim V. 2003.** Gartner. *Service-Oriented Architecture Scenario.* [En línea] 16 de abril de 2003. [Citado el: 17 de febrero de 2008.] AV-19-6751.

**Nuñez, Juan Manuel Barrios. 2003.** *Investigación de la plataforma J2EE y su aplicacion practica.* Chile : Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Departamento de Ciencias de la Computación, 2003.

**OASIS. 2004.** OASIS - Introduction to UDDI: Important Features and Functional Concepts. [En línea] octubre de 2004. [Citado el: 8 de abril de 2008.] <http://uddi.org/pubs/uddi-tech-wp.pdf>.

**Orallo, Enrique Hernández. S/F.** DISCA. *El Lenguaje Unificado de Modelado (UML).* [En línea] S/F. [Citado el: 21 de febrero de 2008.] <http://www.disca.upv.es/enheror/pdf/ActaUML.pdf>.

**Pamela Alejandra Martínez, Alejandra Argueta, Mayra Zavala, Federico Rodríguez, Victor Ordóñez. ERP Crecimiento Planificado de Sistemas de.** [En línea] [Citado el: 10 de 11 de 2007.] [http://maestros.unitec.edu/~carias/ads/documents/ERP\\_VPERIISEM2001.pdf](http://maestros.unitec.edu/~carias/ads/documents/ERP_VPERIISEM2001.pdf).

**Parra, José David. 2006.** MSDN. *Hacia una Arquitectura Empresarial basada en Servicios.* [En línea] 21 de mayo de 2006. [Citado el: 18 de febrero de 2008.] <http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art143..>

- Perry, Dewayne E. y Wolf, Alexander L. 1992.** *Foundations for the study of software architecture*. s.l. : ACM SIGSOFT Software Engineering Notes, 1992. 17(4), pp. 40–52.
- Perry, Dewayne. 1997.** *Software Architecture and its relevance for Software Engineering*. s.l. : Coord'97, 1997.
- PostgreSQL. 2008.** PostgreSQL. [En línea] 2008. [Citado el: 25 de abril de 2008.] <http://www.postgresql.org/about/>.
- PostgreSQL. 2005.** postgresql - Chapter 12. Concurrency Control. [En línea] 2005. [Citado el: 24 de abril de 2008.] <http://www.postgresql.org/docs/8.1/static/mvcc.html>.
- Pressman, Roger S. 2002.** *Ingeniería de Software. Un enfoque practico*. s.l. : McGraw.Hill/Interamericana de España., 2002. 5.
- Reinoso, Carlos Billy. 2006.** MSDN. *MSDN*. [En línea] 26 de junio de 2006. [http://www.microsoft.com/spanish/msdn/arquitectura/roadmap\\_arq/intro.mspx](http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx).
- Reynoso, Carlos Billy. 2006.** MSDN Introducción a la Arquitectura de Software. [En línea] 26 de 6 de 2006. [Citado el: 8 de 2 de 2008.] [http://www.microsoft.com/spanish/msdn/arquitectura/roadmap\\_arq/intro.mspx](http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx).
- . **2005.** MSDN webcast#1. *Seminario de Arquitectura de Software*. [En línea] 2005. [Citado el: 18 de febrero de 2008.] <http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx>.
- . **2005.** MSDN webcast#2. *Profundizando en Estilos de Arquitectura de Software*. [En línea] 2005. [Citado el: 13 de febrero de 2008.] <http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx>.
- . **2005.** MSDN webcast#3. *Arquitectura para distribución y agregación: Services Oriented Architecture (SOA)*. [En línea] 2005. [Citado el: 16 de febrero de 2008.] <http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx>.
- Reynoso, Carlos Billy y Kicillof, Nicolás. 2004.** MSDN Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. [En línea] 2004. [Citado el: 13 de 2 de 2008.] [http://www.microsoft.co.ke/spanish/msdn/arquitectura/roadmap\\_arq/style.aspx](http://www.microsoft.co.ke/spanish/msdn/arquitectura/roadmap_arq/style.aspx).
- . **2004.** MSDN, Lenguajes de Descripción de Arquitectura (ADL). [En línea] UNIVERSIDAD DE BUENOS AIRES, marzo de 2004. [Citado el: 20 de febrero de 2008.] [http://www.microsoft.com/spanish/msdn/arquitectura/roadmap\\_arq/lenguaje.mspx#EKB](http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.mspx#EKB).
- Rumbaugh, James, y otros. 1991.** *Object-oriented modeling and design*. Prentice Hall : Englewood Cliffs, 1991.
- Sanchez, Alvaro Norberto Silva. 2006.** *Logística de Almacenamiento*. Caracas : Tecana American University, 2006. 2.

- Sanz, Laura Bermejo y Monreal, Enrique Gómez. 2007.** Kybele: Eclipse como IDE. *Universidad Rey Juan Carlos*. [En línea] 16 de mayo de 2007. [Citado el: 31 de marzo de 2008.]  
<http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf>.
- Schulte, Roy W. 2002.** Gartner. *Predicts 2003: SOA Is Changing Software*. [En línea] 9 de diciembre de 2002. [Citado el: 17 de febrero de 2008.] AV-18-9758.  
SEI (Software Engineering Institute). *SEI (Software Engineering Institute)*. [En línea] Carnegie Mellon University. [Citado el: 21 de noviembre de 2007.] [www.sei.cmu.edu/architecture/arid.html](http://www.sei.cmu.edu/architecture/arid.html).
- Shaw, Mary y Garlan, David. 1994.** *Characteristics of Higher-Level Languages for Software Architecture*. s.l. : Carnegie Mellon University, 1994. Technical Report CMU-CS-94-210.
- Szyperski, Clemens. 2002.** *Component software: Beyond object-oriented programming*. s.l. : Addison-Wesley Pub Co, 2002. 2da edición.
- . 2000. *Components and Architecture. Software Development*. 2000. online:10.
- Torrico, Gerber Fuentes y Cabrera, Lineth Cintya Camacho. 2007.** Postgrado en Informática: BenchMark sobre Tecnologías Faces. *pgi.umsa.bo*. [En línea] septiembre de 2007. [Citado el: 11 de mayo de 2008.]  
[http://pgi.umsa.bo/enlaces/investigacion/pdf/INGSW3\\_50.pdf?PHPSESSID=ceef241421ecd67ac959046402fb7535](http://pgi.umsa.bo/enlaces/investigacion/pdf/INGSW3_50.pdf?PHPSESSID=ceef241421ecd67ac959046402fb7535).
- Unifé. 2007.** Unifé. *Universidad Femenina del Sagrado Corazón*. [En línea] 10 de abril de 2007. [Citado el: 21 de marzo de 2008.] <http://www.unife.edu.pe/ingenieria/desarrollo.doc>.
- Velasco, Carlos Canal. 2000.** *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software*. Málaga : Universidad de Málaga, 2000.
- Vestal, Steve. 1993.** *A cursory overview and comparison of four Architecture Description Languages*. s.l. : Honeywell Technology Center, 1993.
- W3C. 2007.** W3C - SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation. [En línea] 27 de abril de 2007. [Citado el: 8 de abril de 2008.] <http://www.w3.org/TR/soap12-part1/#intro>.
- Wang, Guijun y Fung, Casey. 2004.** *Architecture paradigms and their influences and impacts on componente-based software systems*. Hawaii : Proceedings of the 37th Hawaii International Conference on System Sciences, 2004.
- Welicki, León. 2007.** MSDN. *Patrones y Antipatrones: una Introducción - Parte II*. [En línea] 2007. [Citado el: 23 de febrero de 2008.]  
[http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ\\_3317.aspx](http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3317.aspx).
- Wolf, Alexander. 1997.** *Succeedings of the Second International Software Architecture Workshop (ISAW-2)*. s.l. : ACM SIGSOFT Software Engineering Notes, 1997. pp. 42-56.