



**INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY**
Campus Ciudad de México

Escuela de Graduados en Ingeniería y Arquitectura

Maestría en Ciencias de la Computación

**IMPORTANCIA DE LA DOCUMENTACION DE LA ARQUITECTURA DE SOFTWARE
PARA EL DESARROLLO DE SISTEMAS : CASO PRACTICO SISTEMA DE
CONCILIACIÓN DE INDEVAL.**

TESIS QUE PRESENTA:

ÁNGEL LUIS RIVERA LANDA

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

Tabla de Contenido

CAPITULO I.....	1
INTRODUCCIÓN.	1
1.1. Introducción.	1
1.2. Planteamiento del Problema.	4
1.3. Objetivo de la investigación.	5
1.4. Preguntas de Investigación.	5
1.5. Hipótesis.....	6
1.6. Delimitación de la Investigación.	6
1.7. Justificación.....	9
1.8. Plan de Trabajo.	12
CAPITULO II.....	13
MARCO TEÓRICO	13
2. Importancia de la Arquitectura de Software.	15
2.1. Definición de Arquitectura de Software.	17
2.2. Enfoques Para la documentación de Arquitecturas de Software.	21
2.2.1. Kruchten “4+1”.....	21
2.2.1.1. Vista Lógica.	22
2.2.1.2. Vista de procesos.	24
2.2.1.3. Vista de desarrollo.	25
2.2.1.4. Vista física.....	28
2.2.1.5. Escenarios.....	29
2.2.1.6. Documentación de las 4+1 vistas.	30
2.2.2. RM-ODP (<i>Reference Model For Open Distributed Processing</i>).	31
2.2.2.1. Punto de vista de la Empresa.....	32
2.2.2.2. Punto de Vista de Información.	32
2.2.2.3. Punto de vista Computacional.	33
2.2.2.4. Punto de vista de Ingeniería.....	33
2.2.2.5. Punto de vista de Tecnología.....	33
2.2.3. SIEMENS (Hofmeister, Nord, and Soni).....	33
2.2.3.1. Vista Conceptual.	34
2.2.3.2. Vista de módulo.	34
2.2.3.3. Vista de Ejecución.	35
2.2.4. SEI Viewtypes.....	35
2.2.5. IEEE 1471-2000.	40
2.2.5.1. Metas y objetivos IEEE1471.....	41
2.2.5.2. Catalogo de Vistas propuesto por el IEEE1471.	43
2.2.5.2.1. Punto de vista Funcional.	43
2.2.5.2.2. Punto de vista de Información.	45
2.2.5.2.3. Punto de vista de Concurrencia.	50
2.2.5.2.4. Punto de vista de Desarrollo.	53
2.2.5.2.5. Punto de vista de Despliegue.	58
2.2.5.2.6. Punto de vista de Operación.	62
2.3. Elección de la Metodología	62
2.3.1. 4+1 Desventajas.	63

2.3.2. RM-ODP Desventajas.....	63
2.3.3. Siemens Desventajas.	64
CAPITULO III	65
CASO PRÁCTICO	65
3. Descripción General del Documento.	65
3.1.1. <i>Guía de lectura para los Stakeholders.</i>	67
3.1.2. <i>Formato de documentación de Vista.</i>	68
3.2. Descripción General del Sistema.	70
3.2.1. Contexto del Sistema.	73
3.2.2. Principios Arquitecturales.	74
3.2.3. Atributos de Calidad.	74
3.3. Mapa de Tecnologías.....	77
3.3.1. Herramientas de Soporte.	77
3.3.2. Herramientas para Pruebas.....	78
3.3.3. Herramientas de Integración.....	78
3.3.4. Motor de Reportes.	79
3.3.5. Herramientas de desarrollo para la capa de presentación.....	80
3.3.6. Servidor de Aplicaciones.	82
3.3.7. Herramientas para la calendarización de tareas.....	83
3.3.8. Lenguajes de Programación.....	84
3.4. Requerimientos Funcionales.....	84
3.4.1. Casos de Uso del Sistema.....	87
3.5. Vista Funcional.....	87
3.5.1. Estructura de Subsistemas.	87
3.5.1.1. Catálogo de Elementos.	88
3.5.1.1.1. Extractor de Información.	88
3.5.1.1.2. Administrador de la Conciliación.	89
3.5.1.1.3. Administrador de Notificaciones.....	89
3.5.1.1.4. Generador de Correos.....	90
3.5.1.1.5. Administrador de Eventos.	90
3.5.1.1.6. Administrador de Catálogos.	91
3.5.1.2. Framework de Aplicación.	92
3.5.1.2.1. CatalogoService.....	93
3.5.1.2.2. BitacoraEventosService.....	94
3.5.1.2.3. NotificacionService.....	94
3.5.1.2.4. ConciliaPartidaService.....	95
3.5.1.2.5. InformacionService.....	96
3.6. Vista de Información.....	96
3.6.1. Modelo Entidad Relación.	96
3.6.2. Catálogo de Elementos.	98
3.7. Vista de Desarrollo.	105
3.7.1. Presentación Inicial.	106
3.7.1.1. Capa de Presentación.....	106
3.7.1.1.1. Principales Reglas.....	109
3.7.1.1.2. Nomenclatura de los componentes.	109
3.7.2. Capa de Negocio.....	110
3.7.2.1. Reglas.....	111
3.7.3. Capa de Acceso a datos.	111
3.7.4. Capa de Persistencia.	112

3.7.4.1. Notación.....	112
3.7.5. Convenciones de Nombrado.....	114
3.8. Vista de Operación.....	116
3.8.1. Instalación del sistema.....	116
3.8.2. Monitoreo de la aplicación.....	118
CAPITULO IV	121
RESULTADOS	121
CAPITULO V.....	125
Conclusiones.....	125
Referencias.....	128

TABLA DE FIGURAS

FIGURA 1. ÁREAS QUE CONFORMAN INDEVAL.....	2
FIGURA 3. NOTACIÓN DE VISTA LÓGICA DEL MODELO 4+1.....	23
FIGURA 4. NOTACIÓN DE LA VISTA DE PROCESOS DEL MODELO 4+1.	25
FIGURA 5. NOTACIÓN DE LA VISTA DE DESARROLLO DEL MODELO 4+1.	27
FIGURA 6. MODELO DE CAPAS DE UN SISTEMA DE SOFTWARE.	28
FIGURA 7. NOTACIÓN DE LA VISTA FÍSICA DEL MODELO 4+1.....	29
FIGURA 8. VISTAS DEL MODELO RMODP.	32
FIGURA 9. VISTAS DEL MODELO DE SIEMENS.	34
FIGURA 10 AGRUPACIÓN DE LAS VISTAS PARA EL MODELO 1471-2000	41
FIGURA 11. EJEMPLO DE UN MODELO DE DATOS ESTÁTICO.	47
FIGURA 12. DIAGRAMA DE FLUJO DE DATOS.	48
FIGURA 13. DIAGRAMA DE CICLO DE VIDA.....	49
FIGURA 14. EJEMPLO DE DIAGRAMA DE ESTADOS PARA UN PROCESADOR DE CONSULTAS.	53
FIGURA 15 MÓDULOS DE UN SISTEMA.	55
FIGURA 16.EJEMPLO DE MODELO DE PLATAFORMA.	59
FIGURA 17.EJEMPLO DE MODELO DE RED	60
FIGURA 18. MOVIMIENTOS DE VALORES EN UNA BÓVEDA.	71
FIGURA 19 PROCESO DE CONCILIACIÓN.....	71
FIGURA 20. DIAGRAMA DE CONTEXTO MIC.	73
FIGURA 21. MODELO DE SEGURIDAD PARA EL SISTEMA DE CONCILIACIÓN.....	77
FIGURA 22 CASOS DE USO DEL SISTEMA.	87
FIGURA 23 PRINCIPALES COMPONENTES DEL MÓDULO DE CONCILIACIÓN.....	88
FIGURA 27. MODELO ENTIDAD RELACIÓN DEL SISTEMA MIC.	97
FIGURA 28. ESTRUCTURA DE LAS FUENTES DE DATOS EXTERNAS.....	104
FIGURA 29. TRANSICIONES DE LAS PRIORIDADES DE UNA NOTIFICACIÓN.	105
FIGURA 30. CICLO DE VIDA DE UNA NOTIFICACIÓN.....	105
FIGURA 31. ESTRUCTURA DE CAPAS DEL MIC.....	106
FIGURA 32. MARCO DE TRABAJO PARA LA CAPA DE PRESENTACIÓN.	108
FIGURA 33.. EJEMPLO DE ESTRUCTURA DE DIRECTORIOS CREADOS CON MAVEN.	118

LISTADO DE TABLAS

TABLA 1. PROPIEDADES DE LA VISTA DE MÓDULO.	36
TABLA 2. PROPIEDADES DE LA VISTA DE COMPONENTES Y CONECTORES.	37
TABLA 3. PROPIEDADES DE LA VISTA DE ASIGNACIÓN.....	37
TABLA 4. DEPENDENCIAS DE SOFTWARE PARA EL NODO DEL SERVIDOR PRIMARIO DE LA FIGURA.....	61
TABLA 5. COMPARACIÓN DE CAPAS ENTRE LAS DIFERENTES PROPUESTAS ANALIZADAS.....	64
TABLA 6. PRINCIPALES MÉTODOS DEL SERVICIO DE CATÁLOGOS.....	93
TABLA 7. PRINCIPALES MÉTODOS DEL SERVICIO DE EVENTOS.....	94
TABLA 8. PRINCIPALES MÉTODOS DEL SERVICIO DE NOTIFICACIÓN.....	95
TABLA 9. PRINCIPALES MÉTODOS DEL SERVICIO DE CONCILIACIÓN.....	96
TABLA 10. PRINCIPALES MÉTODOS DEL SERVICIO DE INFORMACIÓN.....	96
TABLA 11. DESCRIPCIÓN DE CAMPOS PARA C_AREA_INDEVAL.....	98
TABLA 12. DESCRIPCIÓN DE CAMPOS PARA C_GRUPO.....	98
TABLA 13. DESCRIPCIÓN DE CAMPOS PARA C_USUARIO.....	99
TABLA 14. DESCRIPCIÓN DE CAMPOS PARA R_GRUPO_USUARIO.....	99
TABLA 15. DESCRIPCIÓN DE CAMPOS PARA T_NOTIFICACION.....	100
TABLA 16. DESCRIPCIÓN DE LOS CAMPOS DE C_PARAMETROS_MIC.....	101
TABLA 17. DESCRIPCIÓN DE LOS CAMPOS DE T_POSICION.....	102
TABLA 18. DESCRIPCIÓN DE LOS CAMPOS DE T_MOVIMIENTO.....	103
TABLA 19. DESCRIPCIÓN DE LOS CAMPOS DE T_BITACORA_PARTIDA.....	103
TABLA 20. CONVENCIONES DE NOMBRADO DE CÓDIGO.....	116

CAPITULO I

Introducción.

1.1. Introducción.

La arquitectura de software es una disciplina la cual tiene pocos años de conformarse como tal; dentro de esta disciplina existe una parte muy importante que es la documentación de la arquitectura; este trabajo pretende mostrar mediante un caso práctico la importancia de tener documentado un sistema utilizando herramientas como vistas, puntos de vista y perspectivas.

Algunos autores como Len Bass [Bass 03] sugieren ver de forma abstracta las vistas de un sistema tomando como ejemplo el cuerpo humano, donde existen neurólogos, ortopedistas, dermatólogos y cardiólogos enfocados en atender subsistemas muy específicos; estos subsistemas se representan de forma diferente y tienen diferentes propiedades todas ellas relacionadas, las cuales juntas describen la arquitectura de un cuerpo humano.

Otros autores como Perry y Wolf [Perry92] realizan una analogía con la arquitectura de edificios en la cual se necesitan tener diferentes vistas de una edificación, las diferentes vistas incluyen, vista eléctrica, vista hidráulica de elevación e incluso modelos que permitan a los usuarios finales una idea de la estructura general.

El caso práctico a desarrollar se basa en la creación de un sistema de conciliación para el área de custodia de Indeval mostrada en la figura 1, la cual es la única Institución privada

en México que cuenta con autorización de acuerdo a la Ley, para operar como Depósito Central de Valores, una Institución para el Depósito de Valores, es aquella sociedad anónima de capital variable que en general tiene por objeto prestar el servicio de guarda, administración, compensación, liquidación y transferencia de valores.

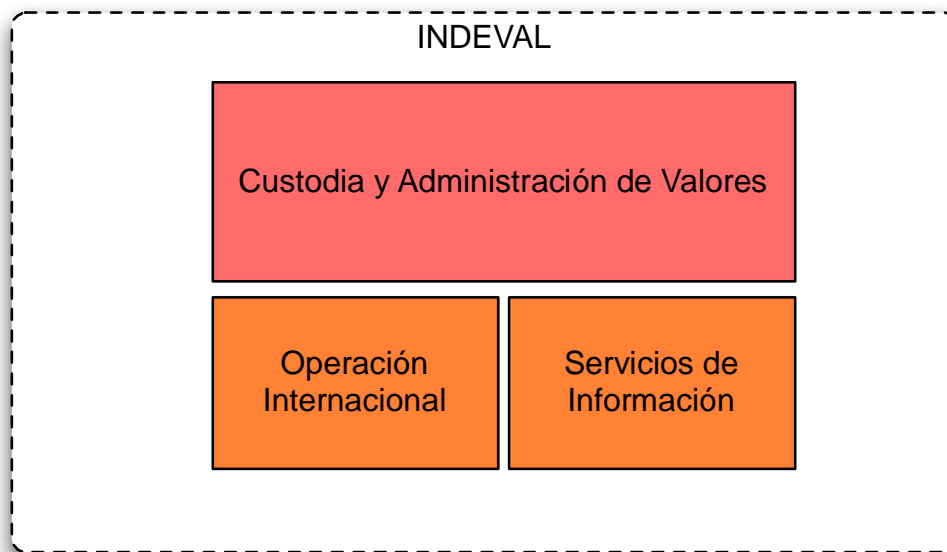


Figura 1. Áreas que conforman Indeval.

Indeval Proporciona los siguientes servicios:

a) Custodia y Administración de Valores

- Guarda física de los valores y/o su registro electrónico en instituciones autorizada para este fin.
- Depósito y retiro físico de documentos de las bóvedas de la institución.
- Ejercicios de derechos en efectivo, en especie y mixtos.

- Transferencia electrónica de valores.
- Transferencia electrónica de efectivo.
- Compensación de operaciones y liquidación DVP.
- Liquidación de operaciones (diversos plazos) para el Mercado de Dinero (directo y reporto) y Mercado de Capitales (operaciones pactadas en la Bolsa).
- Administración de Colaterales.

b) Operación internacional

- Liquidación de operaciones en mercados internacionales
- Administración de derechos patrimoniales de emisiones extranjeras
- Administración de impuestos sobre acciones estadounidenses

c) Servicios de información

- Asignación de códigos ISIN a emisiones
- Servicios a emisoras

Actualmente, Indeval cuenta con varios sistemas que le ayudan a administrar todos los procesos que se llevan a cabo en cada una de sus áreas, estos sistemas han sido actualizados en los últimos años para dar un mejor servicio a las áreas que los utilizan; el esfuerzo que se ha realizado para este cambio ha sido muy grande para la institución ya

que actualmente lleva cuatro años con esta labor y todavía existen áreas que no se han podido atender del todo ya que utilizan sistemas que ya no se adaptan a la forma de operar actual de la institución; una de estas áreas es el área de custodia de la institución que entre otras actividades se realizan las tareas de depósito y retiro físico de documentos en las bóvedas.

De las áreas mencionadas anteriormente, la de custodia y administración de valores es de gran importancia para este trabajo, debido a que la investigación que se pretende realizar involucra los servicios de depósito y custodia de valores de esta área.

Este servicio consiste en la guarda física y/o el registro electrónico de los valores en el S.D. Indeval, quien se hace responsable de dichos valores depositados.

Las características de este servicio son:

- Utilizar el endoso en administración como figura legal, para el depósito y retiro físico de documentos de las bóvedas.
- Inmovilización de documentos.
- Custodia centralizada de todos los valores inscritos en el Registro Nacional de Valores e Intermediarios, que son negociados en mercados financieros, ya sea dentro o fuera de la B.M.V.

1.2. Planteamiento del Problema.

Este trabajo pretende mostrar por qué debería de ser muy importante la producción de un documento de arquitectura de software en la creación de un sistema. Se tratarán de mostrar diferentes enfoques propuestos por la industria de software para la creación de

este documento. No se busca que este documento sea una guía detallada de cada una de las propuestas aquí descritas. En su lugar se pretende resaltar lo esencial de cada una de las propuestas y principales conceptos, todos ellos aterrizados en un caso práctico.

1.3. Objetivo de la investigación.

- Mostrar la importancia que tiene producir la descripción de la arquitectura de software de un sistema.
- Producir la documentación de la arquitectura para el caso práctico del sistema de Indeval.

1.4. Preguntas de Investigación.

- ¿Por qué la documentación de la arquitectura es tan importante para el nuevo sistema de Indeval?
- ¿Qué tipo de documentación es la que Indeval necesita?

1.5. Hipótesis.

Con los conceptos recopilados dentro de la industria de software aplicados en un caso práctico como es el de Indeval, así como la documentación del sistema, resaltando los puntos más importantes del caso, se podrá mostrar la importancia e impacto que un documento de arquitectura de software tiene dentro de un sistema y, con esto, fomentar a que dentro del desarrollo de los sistemas se dedique un tiempo para su realización.

1.6. Delimitación de la Investigación.

Este trabajo de investigación pretende ser descriptivo y correlacional; descriptivo en la parte donde se requiere realizar una investigación para buscar y proponer un conjunto de metodologías para la documentación de la arquitectura de software y correlacional, cuando se usa la tecnología para poder ayudar a Indeval a incrementar la eficiencia de sus procesos en el área de custodia específicamente en el resguardo de valores.

Además de lo anterior, se creará un sistema de software el cual pueda realizar los tres procesos de conciliación correspondientes a las bóvedas de Indeval, Banxico y participantes extranjeros.

El objetivo principal de este trabajo de investigación es el mostrar la importancia de la documentación basándose en un caso práctico, este caso es correspondiente a una nueva aplicación de conciliación para Indeval, dentro de este caso se mostrará el proceso de análisis y desarrollo de la aplicación, sin embargo debido a que existe información de

carácter confidencial, el contenido del caso práctico se enfoca a mostrar de forma general y no particular la aplicación de las técnicas de documentación descritas a lo largo del documento.

Se tienen visualizadas tres etapas dentro del proyecto a desarrollar que satisfacen cada una de las necesidades de conciliación de bóvedas dentro de la institución, cada una de estas etapas contiene una funcionalidad muy específica de la conciliación:

Conciliación de la bóveda de Indeval: Por conciliación se entiende, aquella acción mediante la cual dos posturas encontradas se ponen de acuerdo, y llegan a un arreglo beneficioso para todos, con la ayuda de un tercero neutral e imparcial calificado, denominado: conciliador.

Dentro del área de custodia de Indeval el proceso de conciliación corresponde a verificar que los títulos que se encuentran dentro de la bóveda de Indeval corresponden con los títulos que se encuentran dentro de la bóveda del sistema de Depósito Administración y Liquidación de Valores (Dalí) el cual es el sistema central de liquidación de valores; el proceso debe de conciliar la posición de los títulos así como de los movimientos de depósitos y retiros sobre los mismos.

1. **Conciliación de la bóveda de Banxico:** Este proceso realiza la conciliación de los títulos que se encuentran dentro de la bóveda de Banxico contra los títulos que existen dentro de la bóveda de Dalí, el proceso debe de conciliar la posición de los títulos así como de los movimientos de depósitos y retiros de los mismos.
2. **Conciliación de las bóvedas Internacionales:** Este proceso realiza la conciliación de los títulos que se encuentran las bóvedas de los participantes

extranjeros contra los títulos que existen dentro de la bóveda de Dalí; el proceso debe de conciliar la posición de los títulos así como de los movimientos de depósitos y retiros de los mismos.

Los puntos anteriores se pretenden alcanzar mediante una aplicación *web*, la cual se utilizará de forma interna en el área de custodia; para concluir este trabajo solo se realizará la aplicación *web* con los servicios de conciliación que contemplan el primer punto correspondiente a la conciliación de la bóveda de Indeval debido a que existen los siguientes problemas a tratar antes de realizar las 2 fases subsecuentes:

- Llegar a un acuerdo con los medios de comunicación entre Indeval y Banco de México; actualmente existe un protocolo de comunicación basado en el estándar ISO15022 pero no se han definido los mensajes ni la estructura de la información a intercambiar; el proceso de comunicación con otras entidades es muy largo debido a que las prioridades no están enfocadas al 100% en este proyecto y puede llevar mucho tiempo el definir la comunicación.
- Dentro de Indeval existe un área de internacional donde se lleva a cabo la comunicación con los participantes extranjeros; esta área está renovando sus sistemas y procesos internos y sólo estos son los que pueden tener interacción con la información del extranjero; se debe esperar a que este nuevo sistema sea terminado para analizar la información y solicitar que sea entregada hacia el sistema de conciliación.

La resolución de los problemas anteriores se podrán atender en un trabajo subsecuente debido a que actualmente los requerimientos necesarios que se necesitan atender para su resolución no son conocidos.

Derivado del párrafo anterior los puntos que se resolverán para el caso práctico son los siguientes:

- Análisis de la solución.
- Descripción de la arquitectura de software del sistema.
- Desarrollo de la aplicación web correspondiente a la conciliación con la bóveda de Indeval.
-

1.7. Justificación.

La necesidad de crear sistemas computacionales tiene la finalidad de satisfacer una o varias necesidades dentro de una organización. Estas necesidades derivan en una serie de requerimientos que tienen que ser atendidos para satisfacerlas. No sirve de nada un sistema que no cumple con los atributos de calidad que se especificaron en los requerimientos de los clientes, por lo que, diseñar una correcta arquitectura va a determinar el éxito o fracaso de un sistema de software, en la medida que ésta cumpla o no con sus objetivos.

De acuerdo a [Clements01] existen tres razones por las cuales la arquitectura de software es importante para los grandes y complejos sistemas de software:

- **Es un vehículo para la comunicación entre los *stakeholders*¹.**- La representación del sistema como una abstracción común sirve para que los *stakeholders* puedan usarlo como base para crear un entendimiento mutuo, formar consensos y comunicarse con otros.
- **Es la manifestación temprana de las decisiones de diseño.**- La arquitectura de software del sistema es un artefacto que se usa para detectar de forma temprana los atributos de calidad más importantes del sistema.
- **Es una abstracción reusable y transferible.**- La arquitectura de software constituye un relativamente pequeño modelo de cómo un sistema está estructurado y como sus componentes trabajan en conjunto. Este modelo es transferible a través de otros sistemas y puede ser aplicado a otros sistemas que exhiben similares requerimientos.

Se necesita representar sistemas complejos en una forma manejable y comprensible para los *stakeholders* de negocios hasta los técnicos, por lo que se sugiere atacar el problema desde diferentes direcciones en forma simultánea.

En esta aproximación la descripción de la arquitectura de software es particionada en un conjunto de vistas interrelacionadas, cada una de las cuales describe diferentes aspectos de la arquitectura.

¹ La definición de este término comprende al conjunto de personas involucrados con el sistema y que tienen ciertos intereses con su realización, por tal razón este término será utilizado de esta forma sin traducirlo al idioma español,

Por lo que, de acuerdo a [Rozanski05] una definición de descripción de arquitectura es:

“un conjunto de productos que documentan una arquitectura de una forma tal que los *stakeholders* pueden entenderla y demostrar que la arquitectura cumple con sus requerimientos.”

La disciplina de la Arquitectura de *software* es el puente que existe entre el análisis, el diseño y la elección de una arquitectura adecuada que permitirá aumentar la predictibilidad del producto final y la detección temprana de riesgos.

1.8. Plan de Trabajo.

	ABRIL			MAYO			JUNIO			JULIO			AGOSTO			SEPTIEMBRE			OCTUBRE			NOVIEMBRE			DICIEMBRE		
Introducción	■	■																									
Planteamiento del problema		■	■																								
Preguntas de investigación		■	■																								
Desarrollo de hipótesis		■	■																								
Análisis		■	■	■	■																						
Desarrollo del Marco Teórico						■	■	■																			
Metodología							■	■	■	■																	
--Documentación de la Arquitectura										■	■	■	■	■													
--Diseño de base de datos																■	■										
--Construcción del modelo de datos																	■	■									
--Construcción de servicios																		■	■	■							
Análisis de Resultados																			■	■	■						
Ajustes al documento																						■	■	■	■	■	

CAPITULO II

Marco Teórico

La historia de la arquitectura de software se remonta a la década de los 60's cuando Edsger Dijkstra de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [Dijkstra68]. También sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores. Aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para posteriores investigadores como Frederick P Brooks quien utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario”, Brooks identificaba y razonaba sobre las estructuras de alto nivel y consideraba que el arquitecto es un agente del usuario,[Brooks75].

Otro investigador importante en la década de los 70's fue David Parnas quien demostró que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada.

Parnas desarrolló temas tales como módulos con ocultamiento de información[Parnas72], para 1992 el primer estudio en que aparece la expresión “arquitectura de software”, en el sentido en

que hoy lo conocemos se puede decir que es el de Perry y Wolf [Perry92]; Puede decirse que Perry y Wolf fundaron la disciplina, y su llamamiento fue respondido en primera instancia por los miembros de lo que podría llamarse la escuela estructuralista de Carnegie Mellon: David Garlan, Mary Shaw, Paul Clements, Robert Allen.

Se trata de una práctica de apenas unos doce años de trabajo constante, que en estos momentos experimenta una nueva ola creativa en el desarrollo cabal de sus técnicas en la obra de Rick Kazman, Mark Klein, Len Bass y otros metodólogos en el contexto del SEI.

2. Importancia de la Arquitectura de Software.

Actualmente para el desarrollo de grandes sistemas es muy raro el no incluir un arquitecto de software, y muchas organizaciones tienen un departamento completo responsable de desarrollar una estrategia de arquitectura y visión de la empresa. de acuerdo a [Rozanski05] un arquitecto de *software* es la persona o grupo de personas responsables del diseño, la documentación y llevar a cabo la construcción de una arquitectura que conoce las necesidades de los *stakeholders*. existe muy poco consenso sobre lo que implica realmente el trabajo:

- ¿Quiénes son los clientes?
- ¿A quién se debe de rendir cuentas?
- ¿Cuál es nuestra participación una vez que el diseño de la arquitectura ha sido completado?
- ¿Donde están los límites entre requerimientos, arquitectura y diseño?

Esta situación hace las cosas muy difíciles para los nuevos arquitectos designados y causa problemas incluso para los más experimentados.

El punto clave a desarrollar como parte del trabajo de un arquitecto de software es producir una descripción de la arquitectura de los sistemas; este documento es el medio con el cual se da a conocer la arquitectura de un sistema, ayuda a demostrar que se han conocido los intereses de los usuarios.

Los puntos clave para crear una descripción de la arquitectura exitosa de acuerdo a [Rozanski05] son:

- Identificar a los *stakeholders* y trabajar con ellos para crear una arquitectura que responda a su complejidad, detectar los requerimientos de los mismos que por lo general son contrapuestos y que frecuentemente se requiere que atiendan a necesidades en conflicto.
- Utilizar vistas y puntos de vista para ayudar a definir y documentar la estructura de la arquitectura, hacer manejable un problema enorme.
- Usar perspectivas para ayudar a comprender los atributos de calidad del sistema tales como el desempeño, disponibilidad, y seguridad—y así asegurar que el sistema cumple sus metas no funcionales.

Comúnmente una arquitectura de software se documenta a través de un conjunto de vistas, en donde cada vista representa un aspecto o comportamiento particular del sistema. Algunos de los artículos de mayor relevancia que abordan el tema del uso de vistas son el conocido, “Modelo de 4+1 vistas de la arquitectura de software” de Philippe B. Kruchten, y el de Robert L. Nord et al. titulado, “La arquitectura de software en aplicaciones industriales”. Ambos artículos fueron publicados en el año de 1995; El primero es el más conocido, quizás esto se deba a que la propuesta de Kruchten es parte fundamental de la metodología del Proceso

Unificado, que en la actualidad es una de las metodologías que goza de cierto grado de popularidad.

2.1. Definición de Arquitectura de Software.

Tratar de definir el término arquitectura de software es un poco complicado ya que existen diferentes definiciones de las cuales ninguna es ampliamente aceptada por la industria de software.

- La arquitectura de software es definida por la práctica recomendada como la organización de un sistema embebido en sus componentes, sus relaciones con otros y su entorno y los principios que gobiernan su diseño y evolución. [ANSI/IEEE 2000]
- La arquitectura de software de un programa o sistema de cómputo es la estructura o estructuras del sistema, los cuales comprenden los elementos de software, las propiedades externas visible de estos elementos y las relaciones entre ellos.[Bass03]
- La arquitectura de software trata con el diseño, estructura e implementación de alto nivel del software, es el resultado de reunir un cierto número de elementos arquitectónicos en ciertas formas bien seleccionadas para satisfacer las funciones principales y requerimientos de rendimiento del sistema, así como algunos otros requerimientos no funcionales, tales como la fiabilidad, escalabilidad, portabilidad y disponibilidad [Kruchten95].
- La arquitectura de software [Clements96] 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.

Revisando las definiciones anteriores nos podemos dar cuenta que la noción clave de la arquitectura es la organización (un concepto cualitativo o estructural). Ante el número y variedad de definiciones existentes de arquitectura de software, Mary Shaw y David Garlan [Shaw96] proporcionaron una sistematización, explicando las diferencias entre definiciones en función de distintas clases de modelos:

- 1) **Modelos estructurales:** Sostienen que la arquitectura de software está integrada por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes.
- 2) **Modelos de *framework*²:** Son similares a la vista estructural, pero su énfasis primario radica en la estructura (usualmente una sola) coherente del sistema completo, en vez de concentrarse en su composición.
- 3) **Modelos dinámicos:** Enfatizan la cualidad conductual de los sistemas. “Dinámico”, puede referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos.
- 4) **Modelos de proceso:** Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción.
- 5) **Modelos funcionales:** Una minoría considera la arquitectura como un conjunto de componentes funcionales, organizados en capas que proporcionan servicios hacia arriba. Es tal vez útil pensar en esta visión como un marco de trabajo particular.

² El uso de Framework y marco de trabajo se usa de forma indistinta dentro del documento y los dos términos se refieren al mismo concepto.

Existen unos cuantos organismos de estándares (ISO *International Organization for Standardization*, CEN *Comité Européen de Normalisation*, IEEE *Institute of Electrical and Electronics Engineers*, OMG *Object Management Group*) que han codificado diferentes aspectos de la arquitectura de software, con el objetivo de homogeneizar la terminología, los modelos y los procedimientos.

Tanto los marcos arquitectónicos como las metodologías de modelado de los organismos acostumbran ordenar las diferentes perspectivas de una arquitectura en términos de vistas (*views*). La mayoría de los marcos de trabajo y estrategias reconoce entre tres y seis vistas, las cuales se usan para definirla como un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado.

Perry y Wolf 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. A diferencia de los patrones de diseños, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo.

Rozanski [Rozanski05] propone un catálogo de estilos que son desde su punto de vista particularmente relevantes:

- **Tuberías y Filtros.-** Es un estilo muy sencillo, en el cual existe un elemento denominado filtro cuyo propósito es el de procesar flujos de datos, con instancias de este tipo conectados por simples conectores conocidos como tuberías.
- **Cliente/Servidor.-** Es un estilo ampliamente utilizado. Define una estructura de sistema comprendida por dos tipos de elementos: un servidor que provee uno o más servicios vía una interfaz bien definida y un cliente que utiliza los servicios como parte de su operación. El cliente y el servidor típicamente se asume que residen en diferentes máquinas en una red, lo cual no es un requerimiento.
- **Punto a Punto.-** Frecuentemente referido como P2P. Este estilo define un sencillo tipo de elemento peer y un sencillo tipo de conector que es una conexión de red, el principio de este estilo radica en que cualquier par se puede comunicar directamente con cualquier otro sin necesidad de usar un servidor central. Cada par se puede comportar como cliente o servidor.
- **Implementación en Capas.-** En este estilo se identifica un sencillo tipo de elemento de sistema: la capa. Este estilo organiza un sistema como una pila de capas, con cada capa provee servicios a la capa superior y solicita servicios a la capa inferior. Estas capas están ordenadas por un nivel de abstracción, desde la capa más abstracta en el nivel superior hasta las capas más concretas en los niveles inferiores.
- **Publicador/Suscriptor.-** Este estilo define un elemento que es el publicador, el cual crea información de interés a cualquier número de elementos del sistema (suscriptores) que pueden consumirla.

2.2. Enfoques Para la documentación de Arquitecturas de Software.

2.2.1. Kruchten “4+1”.

Esta aproximación sugiere el uso de un conjunto de escenarios para ilustrar como un conjunto de vistas trabajan en conjunto, en la figura 2 se muestra este modelo.

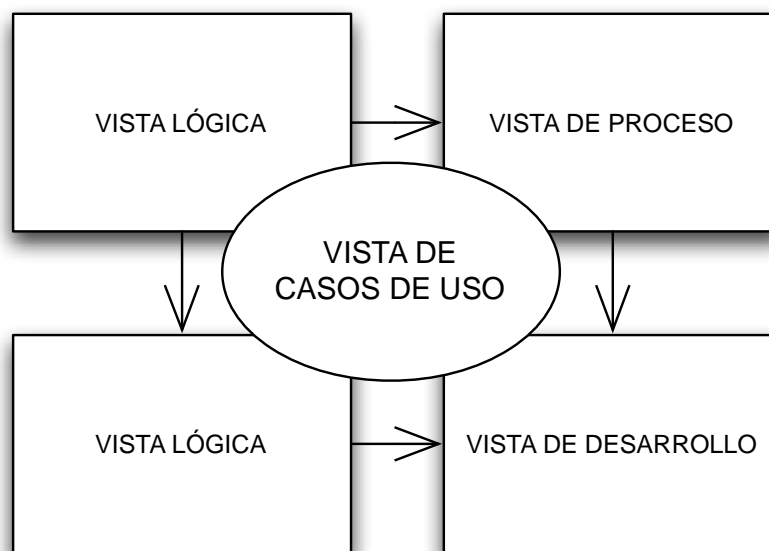


Figura 2. modelo 4+1.

La vista lógica describe el modelo de objetos del diseño cuando se usa un método de diseño orientado a objetos. Para diseñar una aplicación muy orientada a los datos, se puede usar un enfoque alternativo para desarrollar algún otro tipo de vista lógica, tal como diagramas de entidad-relación.

- La vista de procesos describe los aspectos de concurrencia y sincronización del diseño.
- La vista física describe el mapeo del software en el hardware y refleja los aspectos de distribución.
- La vista de desarrollo describe la organización estática del software en su ambiente de desarrollo.

Cada vista se describe en lo que llamamos “diagrama” (*blueprint*) que usa su notación particular. Los arquitectos también pueden usar estilos de arquitectura para cada vista, y por lo tanto hacer que coexistan distintos estilos en un mismo sistema.

El modelo de 4+1 vistas es bastante genérico: se puede usar otra notación y herramientas que las aquí descritas, así como también otros métodos de diseño, especialmente para las descomposiciones lógica y de procesos.

2.2.1.1. Vista Lógica.

La arquitectura lógica apoya principalmente los requisitos funcionales –lo que el sistema debe brindar en términos de servicios a sus usuarios. El sistema se descompone en una serie de abstracciones clave, tomadas (principalmente) del dominio del problema en la forma de objetos o clases de objetos. Aquí se aplican los principios de abstracción, encapsulamiento y herencia. Esta descomposición no sólo se hace para potenciar el análisis funcional, sino también sirve para identificar mecanismos y elementos de diseño comunes a diversas partes del sistema.

Se usa el enfoque de *Booch/Rational* para representar la arquitectura lógica, mediante diagramas de clases y plantillas de clases. Un diagrama de clases muestra un conjunto de clases y sus relaciones lógicas. [Krushten95]

Notación. La notación para la vista lógica es la que se muestra en la figura 3 y se deriva de la notación de [Booch93]. Esta se simplifica considerablemente de tal modo de tener en cuenta solamente los elementos relevantes para la arquitectura. En particular, los numerosos adornos disponibles son bastante inútiles para este nivel de diseño.

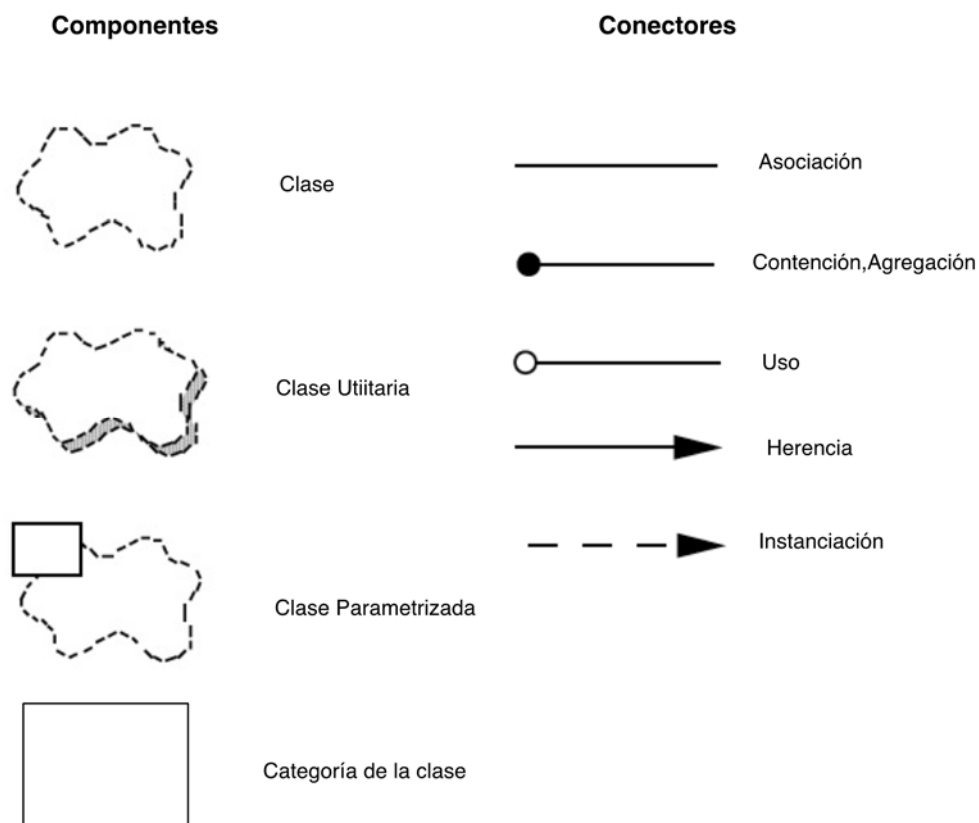


Figura 3. Notación de vista lógica del modelo 4+1.

Estilo. El estilo usado para la vista lógica es el estilo de orientación a objetos. La principal guía para el diseño de la vista lógica es el intentar mantener un modelo único y coherente de objetos

a lo largo de todo el sistema, para evitar la especialización prematura de las clases y mecanismos particulares o de un procesador.

2.2.1.2. Vista de procesos.

La arquitectura de procesos toma en cuenta algunos requisitos no funcionales tales como el performance y la disponibilidad. Se enfoca en asuntos de concurrencia, distribución, integridad del sistema y de tolerancia a fallas. La vista de procesos también especifica en cuál hilo de control se ejecuta efectivamente una operación de una clase identificada en la vista lógica.

La arquitectura de procesos se describe en varios niveles de abstracción, donde cada nivel se refiere a distintos intereses. El nivel más alto la arquitectura de procesos puede verse como un conjunto de redes lógicas de programas comunicantes (llamados “procesos”) ejecutándose en forma independiente, y distribuidos a lo largo de un conjunto de recursos de hardware conectados mediante un bus, una LAN o WAN. Múltiples redes lógicas pueden usarse para apoyar la separación de la operación del sistema en línea del sistema fuera de línea, así como también para apoyar la coexistencia de versiones de software de simulación o de prueba. Un proceso es una agrupación de tareas que forman una unidad ejecutable. Los procesos representan el nivel al que la arquitectura de procesos puede ser controlada tácticamente (i.e., comenzar, recuperar, reconfigurar, y detener). Además, los procesos pueden replicarse para aumentar la distribución de la carga de procesamiento, o para mejorar la disponibilidad.

Notación. La notación que se usa para la vista de procesos es la que se muestra en la figura 4 y se expande de la notación originalmente propuesta por Booch para las tareas de Ada y se centra solamente en los elementos arquitectónicamente relevantes.

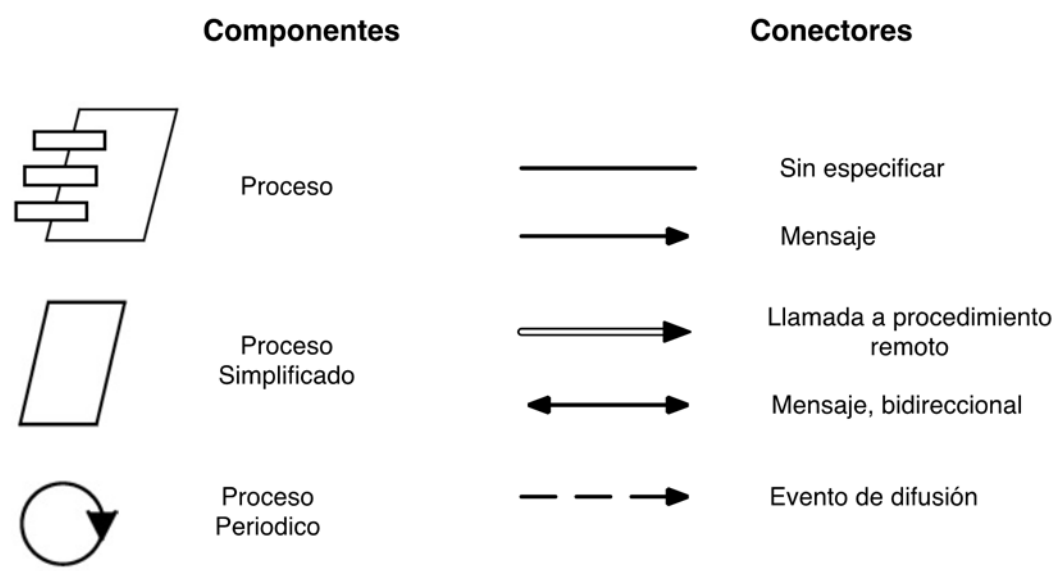


Figura 4. Notación de la vista de procesos del modelo 4+1.

Estilo. Varios estilos podrán servir para la vista de procesos. Por ejemplo, tomando la taxonomía de Garlan y Shaw [Garlan93] tenemos: tubos y filtros, o cliente/servidor, con variantes de varios clientes y un único servidor o múltiples clientes y múltiples servidores. Para sistemas más complejos, podemos usar un estilo similar a la forma de agrupación de procesos del sistema ISIS descrito por Kenneth Birman con otra notación y otras herramientas.

2.2.1.3. Vista de desarrollo.

La vista de desarrollo se centra en la organización real de los módulos de software en el ambiente de desarrollo del *software*. El software se empaqueta en partes pequeñas –bibliotecas de programas o subsistemas– que pueden ser desarrollados por uno o un grupo pequeño de desarrolladores. Los subsistemas se organizan en una jerarquía de capas, cada una de las cuales brinda una interfaz estrecha y bien definida hacia las capas superiores.

La vista de desarrollo tiene en cuenta los requisitos internos relativos a la facilidad de desarrollo, administración del software, reutilización y elementos comunes, y restricciones impuestas por las herramientas o el lenguaje de programación que se use. La vista de desarrollo apoya la asignación de requisitos y trabajo al equipo de desarrollo, y apoya la evaluación de costos, la planificación, el monitoreo de progreso del proyecto, y también como base para analizar re uso, portabilidad y seguridad. Es la base para establecer una línea de productos.

La vista de desarrollo de un sistema se representa en diagramas de módulos o subsistemas que muestran las relaciones exporta e importa. La arquitectura de desarrollo completa sólo puede describirse completamente cuando todos los elementos del software han sido identificados. Sin embargo, es posible listar las reglas que rigen la arquitectura de desarrollo – partición, agrupamiento, visibilidad – antes de conocer todos los elementos.

Notación. Tal como se muestra en la Figura 5, usamos una variante de la notación de Booch limitándonos a aquellos elementos relevantes para la arquitectura.

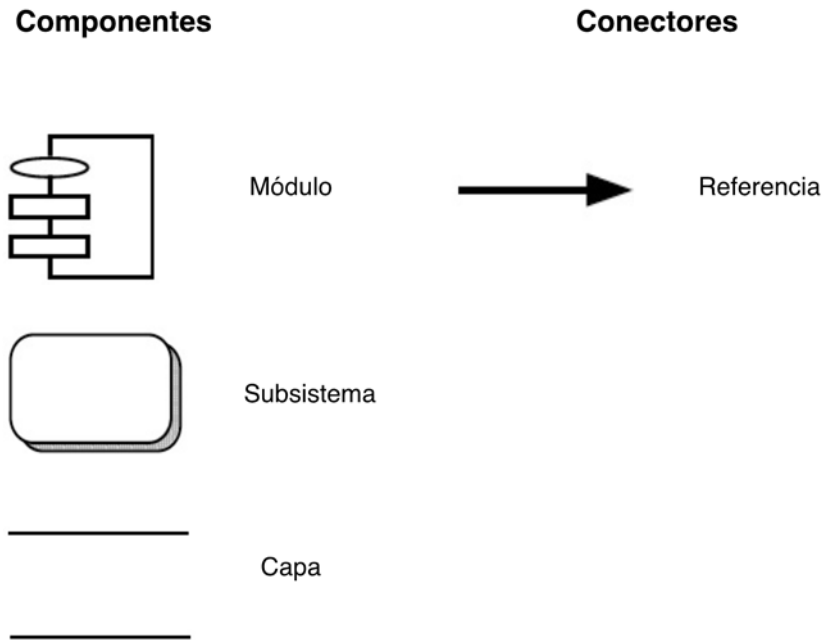


Figura 5. Notación de la vista de desarrollo del modelo 4+1.

Estilo para la vista de desarrollo. Recomendamos adoptar el estilo de capas mostrado en la figura 6 para la vista de desarrollo, definido en 4 a 6 niveles de subsistemas. Cada capa tiene una responsabilidad bien definida. La regla de diseño es que un subsistema en una cierta capa sólo puede depender de subsistemas que estén o bien en la misma capa o en capas inferiores, de modo de minimizar el desarrollo de complejas redes de dependencias entre módulos y permitir estrategias de desarrollo capa por capa.

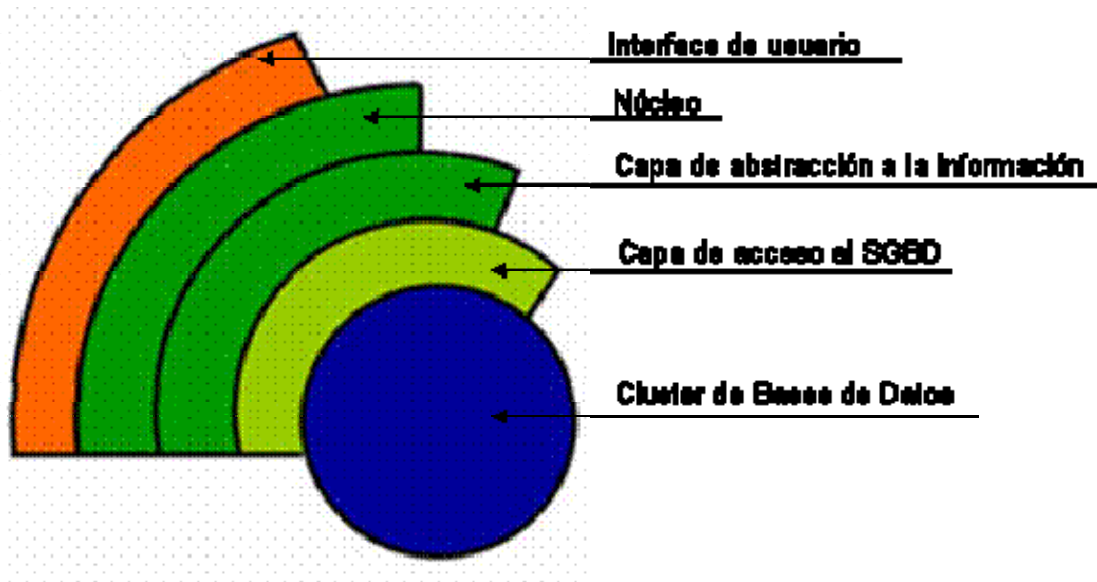


Figura 6. Modelo de capas de un sistema de software.

2.2.1.4. Vista física.

La arquitectura física toma en cuenta primeramente los requisitos no funcionales del sistema tales como la disponibilidad, confiabilidad (tolerancia a fallas), desempeño, y escalabilidad. El software ejecuta sobre una red de computadores o nodos de procesamiento (o tan solo nodos). Los variados elementos identificados –redes, procesos, tareas y objetos – requieren ser mapeados sobre los variados nodos; Esperamos que diferentes configuraciones puedan usarse: algunas para desarrollo y pruebas, otras para emplazar el sistema en varios sitios para distintos usuarios. Por lo tanto, el mapeo del software en los nodos requiere ser altamente flexible y tener un impacto mínimo sobre el código fuente en sí.

Notación. Los diagramas físicos pueden tornarse muy confusos en grandes sistemas, y por lo tanto toman diversas formas, con o sin el mapeo de la vista de procesos, la notación propuesta para esta vista se muestra en la figura 7.

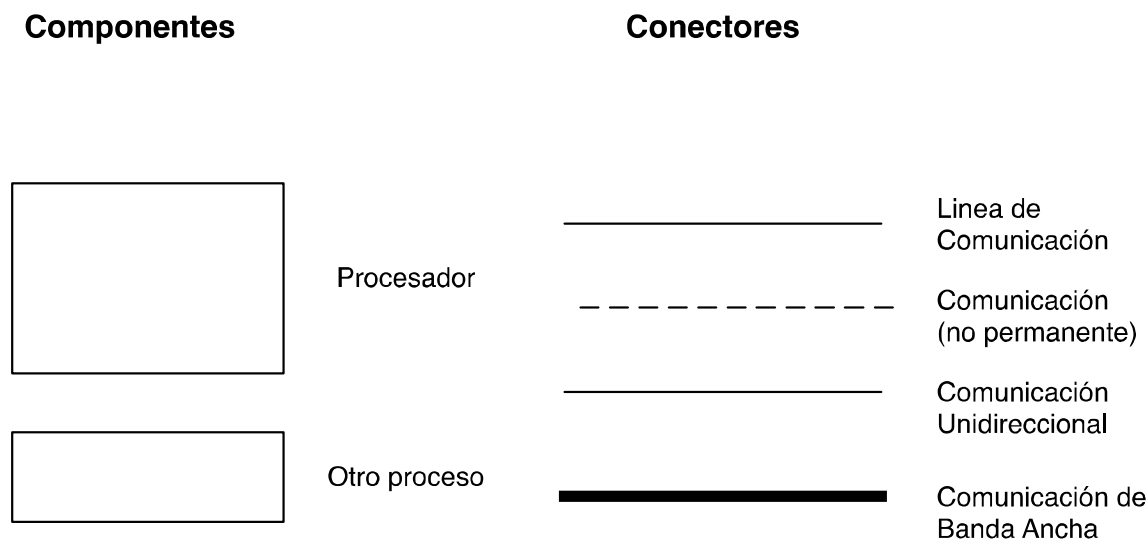


Figura 7. Notación de la vista física del modelo 4+1.

2.2.1.5. Escenarios.

Los elementos de las cuatro vistas trabajan conjuntamente en forma natural mediante el uso de un conjunto pequeño de escenarios relevantes –instancias de casos de uso más generales– para los cuales describimos sus *scripts* correspondientes (secuencias de interacciones entre objetos y entre procesos) tal como lo describen Rubin y Goldberg [Rubin92]. Los escenarios son de alguna manera una abstracción de los requisitos más importantes.

Su diseño se expresa mediante el uso de diagramas de escenarios y diagramas de interacción de objetos.

Esta vista es redundante con las otras (y por lo tanto “+1”), pero sirve a dos propósitos principales:

- Como una guía para descubrir elementos arquitectónicos durante el diseño de arquitectura tal como lo describiremos más adelante.
- Como un rol de validación e ilustración después de completar el diseño de arquitectura, en el papel y como punto de partida de las pruebas de un prototipo de la arquitectura.

Notación para escenarios. La notación es muy similar a la vista lógica para los componentes, pero usa los conectores de la vista de procesos para la interacción entre objetos. Nótese que las instancias de objetos se denotan con líneas sólidas.

2.2.1.6. Documentación de las 4+1 vistas.

La documentación producida durante el diseño de la arquitectura se captura en dos documentos:

- Un documento de arquitectura del Software, cuya organización sigue las “4+1” vistas
- Un documento de Guías del Diseño del Software, que captura (entre otras cosas) las decisiones de diseño más importantes que deben respetarse para mantener la integridad de la arquitectura del sistema.

Historial de Cambios
Tabla de contenidos
Lista de Figuras
1. Alcances
2. Referencias
3. Arquitectura de Software
4. Objetivos de la arquitectura y Restricciones
5. Arquitectura de la vista Lógica
6. Arquitectura de la vista de Procesos
7. Arquitectura de la vista de desarrollo
8. Arquitectura de la vista Física
9. Escenarios
10. Tamaño y performance
11. Calidad
Apéndices
A. Acrónimos y abreviaciones
B. Definiciones
C. Principios de diseño

2.2.2. RM-ODP (*Reference Model For Open Distributed Processing*).

Lo que *RM-ODP* proporciona es un marco de referencia mediante el cual poder examinar, describir y especificar un sistema desde distintas perspectivas, denominadas puntos de vista. Cada uno de estos puntos de vista trata de satisfacer a una audiencia distinta, cada una interesada en aspectos diferentes del sistema. Y asociado a cada uno de los puntos de vista se define un lenguaje especializado, que recoge el vocabulario y la forma de expresarse de la audiencia concreta a la que se dirige.

RM-ODP define cinco puntos de vista genéricos como se muestran en la figura 8, y que considera básicos a la hora de definir un sistema:

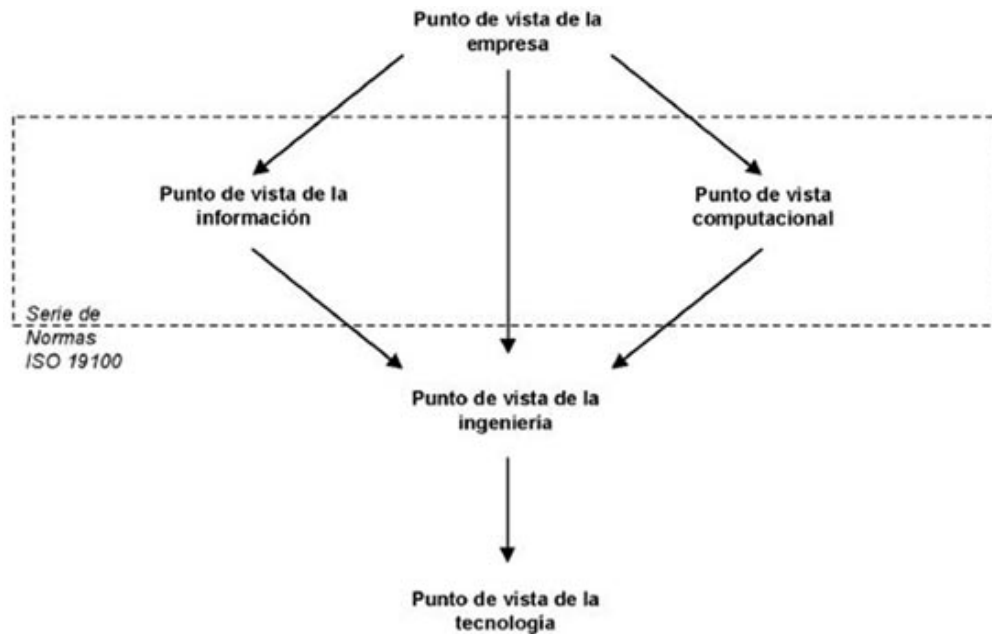


Figura 8. Vistas del modelo RMODP.

2.2.2.1. Punto de vista de la Empresa.

Describe los requisitos desde la perspectiva del propio negocio, así como la manera en la que pretende satisfacerlos. Se centra en la finalidad, alcance, entorno y políticas que rigen las actividades del sistema especificado, dentro de la organización de la que forma parte.

Los modelos de objetos de negocio y políticas de empresa (permisos, prohibiciones y obligaciones), debe de ser comprensible para los clientes y usuarios y facilitar la validación de la arquitectura de software respecto a las necesidades de la empresa.

2.2.2.2. Punto de Vista de Información.

Describe el tipo de información que va a manejar el sistema, así como la estructura de los datos y sus posibles valores. Se centra en las clases de información tratadas por el

sistema, su semántica, y las restricciones impuestas sobre la utilización e interpretación de dicha información.

2.2.2.3. Punto de vista Computacional.

Describe la funcionalidad que ha de ofrecer el sistema, así como su descomposición y organización funcional. Para ello trata de describir el sistema como un conjunto de objetos que interactúan entre sí, definidos mediante interfaces.

2.2.2.4. Punto de vista de Ingeniería.

Describe la infraestructura necesaria para soportar el procesamiento distribuido del sistema, así como la forma de distribución de los datos y operaciones que permitan al sistema proporcionar la funcionalidad requerida.

2.2.2.5. Punto de vista de Tecnología.

Describe la tecnología que soportará el sistema en base a la infraestructura de hardware, software y comunicaciones que permita el procesamiento y la funcionalidad necesaria, así como la representación y distribución de los datos.

2.2.3. SIEMENS (Hofmeister, Nord, and Soni).

Christine Hofmeister, Robert Nord and Dilip Soni como parte de su trabajo en Siemens desarrollaron un conjunto de vistas arquitecturales mostrados en la figura 9 basados en la forma en que los equipos de desarrollo de software que operaban.

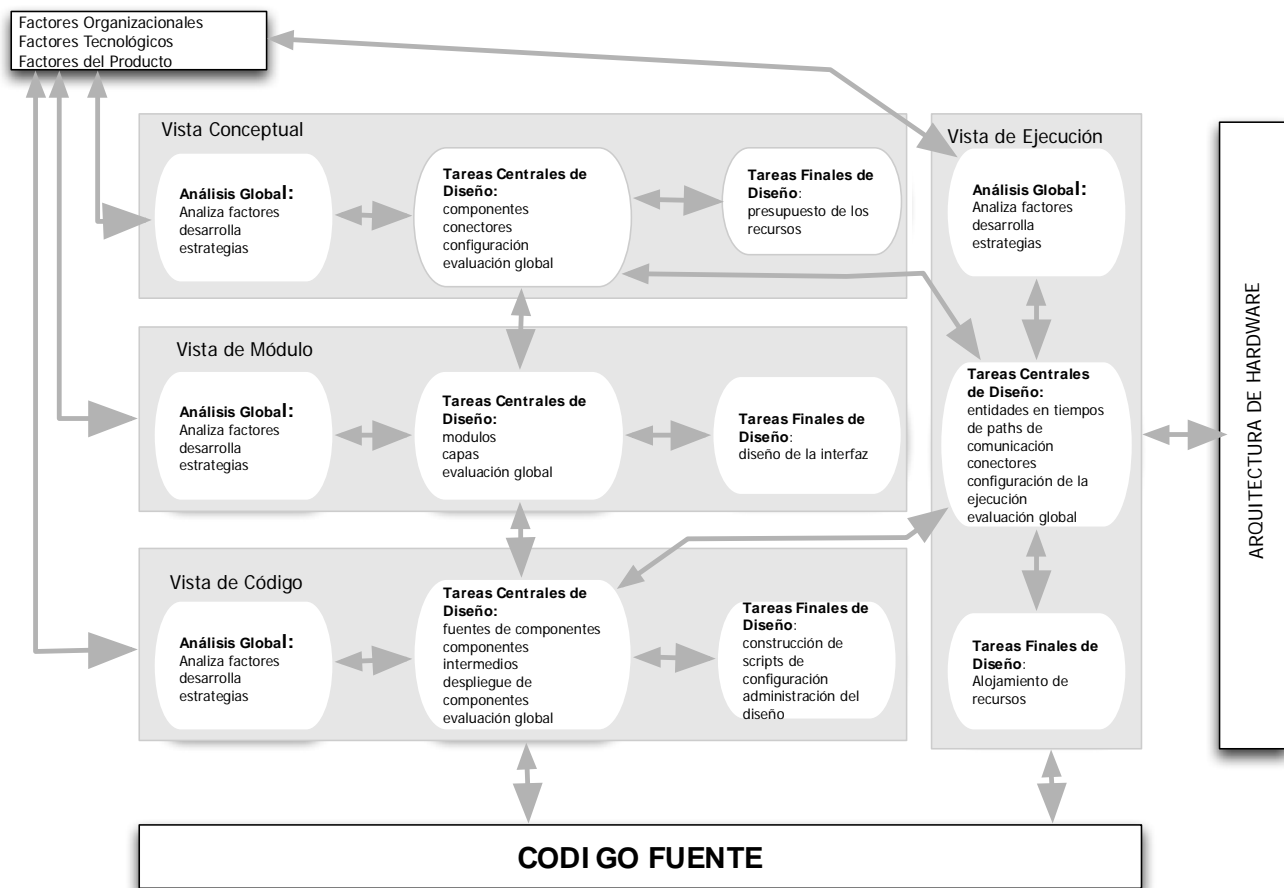


Figura 9. Vistas del modelo de SIEMENS.

2.2.3.1. Vista Conceptual.

Muestra la estructura funcional del sistema la cual define un conjunto de componentes conceptuales ligados por medio de un conjunto de conectores.

2.2.3.2. Vista de módulo.

Muestra las estructuras completas de los subsistemas y módulos que pueden ser realizados dentro del sistema, sus interfaces expuestas por los módulos, la interdependencia de los módulos y cualquier capa de restricciones en la estructura.

2.2.3.3. Vista de Ejecución.

Muestra la estructura del sistema en tiempo de ejecución en término de procesos. Hilos, elementos de comunicación entre procesos.

2.2.4. SEI Viewtypes.

El SEI en su propuesta[*Stafford 02*] define tres categorías denominadas tipos de vista, en las que prácticamente cualquier vista, dependiendo del tipo de información que contenga, puede pertenecer a una de estas categorías. Los tipos de vista pueden ser:

- **Vista de módulo.** Describe cómo el sistema es estructurado en un conjunto de unidades de código.
- **Vista de conectores y componentes.** Describe cómo el sistema es estructurado en un conjunto de elementos que están en tiempo de ejecución así como su interacción.
- **Vista de asignación.** Describe la relación entre las unidades de software y los elementos del entorno como hardware, sistemas de archivos o la organización de los equipos de desarrollo de software.

La documentación de las vistas se realiza a través de lo que se denomina “paquetes de vista”.

Una vista puede llegar a contener gran cantidad de elementos, ocasionando en el personal involucrado dificultad en su interpretación. La solución a esto es utilizar paquetes de vista, los cuales contienen un número reducido de elementos, logrando así una mejor comprensión ya

que solo se muestra un fragmento particular del sistema. De esta manera, una vista se descompone en uno o más paquetes de vista.

Elementos relaciones y propiedades de la vista de modulo se describen en la tabla 1:

Elementos	El elemento de la vista de modulo es un módulo, el cual es una unidad funcional que implementa un conjunto de responsabilidades.
Relaciones	<p>Las relaciones mostradas en una vista de modulo pueden ser algo como:</p> <ul style="list-style-type: none"> • Es parte de • Depende de • Es un <p>A es parte de B, se define como una parte de un todo, una relación entre A y B</p> <p>A depende de B define una relación de dependencia entre A y B.</p> <p>A es un B define una relación de generalización entre un modulo mas especifico y un modulo mas general donde el padre es B y su hijo A</p>
Propiedades de los elementos	<p>Propiedades de un modulo incluyen:</p> <p>Nombre</p> <p>Responsabilidades del modulo</p> <p>Visibilidad de las interfaces de los módulos</p> <p>Información de implementación</p>
Propiedades de las relaciones	

Tabla 1. Propiedades de la vista de módulo.

Los elementos relaciones y propiedades de la vista de componentes y conectores se muestran en la tabla 2:

Elementos	Componentes: Principales unidades de proceso y almacenamiento de datos. Conectores: Mecanismos de interacción.
Relaciones	Adjuntos: define la topología del sistema indicando cuales conectores de los componentes están asociados con otros conectores pertenecientes a otros componentes.
Propiedades de los elementos	Componente: Nombre Tipo Puertos Conector: Nombre Tipo Roles
Propiedades de las relaciones	

Tabla 2. Propiedades de la vista de componentes y conectores.

Elementos relaciones y propiedades de la vista de asignación:

Elementos	Elementos de software y de entorno
Relaciones	Alojado en
Propiedades de los elementos	Los elementos de software tienen propiedades requeridas. Un elemento de entorno tiene propiedades que provee las cuales necesitan ser ajustadas.
Propiedades de las relaciones	

Tabla 3. Propiedades de la vista de asignación.

Es importante señalar que cada tipo de vista viene acompañado de un conjunto predefinido de estilos, de esta forma los arquitectos pueden hacer uso de éstos para documentar las vistas. De acuerdo a [Shaw96], un estilo arquitectónico es una descripción de los elementos, conectores, topología, y un conjunto de restricciones sobre la interacción de los elementos. El uso de estilos beneficia el cumplimiento de las cualidades no funcionales de un sistema; en otras

palabras, promueve la satisfacción de los intereses definidos por parte del personal involucrado en el proyecto.

El SEI recomienda contar con una guía de estilos que contenga entre otros aspectos, la descripción relevante del estilo, elementos, relaciones, propiedades, situaciones en las que no es recomendable aplicarlo, circunstancias en las que se recomienda usar el estilo, y posibles enfoques analíticos que el arquitecto puede utilizar. Para la elaboración de la guía de estilos se puede tomar como referencia el informe técnico realizado por Mark Klein y Rick Kazman [Klein] titulado, “Estilos Arquitectónicos Basados en Atributos”. En éste, los autores proponen un marco de trabajo para llevar a cabo un razonamiento cualitativo o cuantitativo de los atributos de calidad presentes en un estilo arquitectónico. En la parte final del informe, los autores presentan una serie de ejemplos que describen el uso del marco de trabajo con los atributos de calidad: desempeño, facilidad de modificación y disponibilidad. En caso de no encontrar en la literatura algún estilo que satisfaga las necesidades del arquitecto, es totalmente valido elaborar estilos propios y agregarlos a la guía.

La documentación de las vistas se realiza a través de lo que se denomina “paquetes de vista”. Una vista puede llegar a contener gran cantidad de elementos, ocasionando en el personal involucrado dificultad en su interpretación. La solución a esto es utilizar paquetes de vista, los cuales contienen un número reducido de elementos, logrando así una mejor comprensión ya que solo se muestra un fragmento particular del sistema.

Para seleccionar las vistas a documentar, se sigue un procedimiento basado con respecto a las estructuras que se encuentran presentes de manera inherente en el sistema a construir, y en los intereses primarios del personal involucrado. El procedimiento consta de los siguientes pasos:

- 1) **Elaborar una lista de vistas candidatas.** En este paso se elabora una tabla con la siguiente información, en las columnas se enumera el conjunto de posibles vistas a documentar, mientras que en las filas se enumera el personal involucrado. Posteriormente en cada una de las celdas se especifica el grado de información que requiere cada una de las personas involucradas en el proyecto. Los valores posibles para las celdas pueden ser: requerido a detalle, de manera general, o ninguno. Este paso concluye una vez que se han seleccionado las vistas de mayor interés por parte de las personas involucradas.
- 2) **Combinar las vistas.** Puede que las vistas elegidas en el paso anterior sean imprácticas de documentar debido al número de vistas seleccionadas, en este paso se reduce la lista de vistas de una manera que pueda ser manejable por el arquitecto. La reducción se lleva a cabo combinando varias vistas, de este modo una vista combinada muestra la información nativa de dos o más vistas separadas.
- 3) **Priorizar las vistas.** En este paso, el arquitecto debe tener el conjunto mínimo de vistas que satisfacen los intereses del personal involucrado. Después, en conjunto con el administrador del proyecto se procede a priorizar cada una de las vistas resultantes.

Una vez que las vistas se han seleccionado y priorizado, se inicia la documentación de éstas. El SEI cuenta con una plantilla [SEI06] que se puede utilizar de referencia para este propósito. De acuerdo al SEI, la documentación de una arquitectura debe contener los siguientes apartados:

- **Presentación primaria.** Muestra los elementos y sus relaciones entre sí, usualmente se representa de manera gráfica.
- **Catálogo de elementos.** Contiene los detalles de éstos, sus propiedades e interfaces.
- **Diagrama de contexto.** Muestra la relación entre el sistema o porción de éste y su entorno.
- **Guía de variabilidad.** Muestra los posibles puntos de variación en caso de que las vistas sean modificadas.
- **Antecedentes de la arquitectura.** Explica la justificación de la arquitectura así como los supuestos, y los resultados de los análisis realizados.
- **Otra información.** En esta sección se incluyen prácticas y políticas de la organización.
- **Paquetes de vista relacionados.** Básicamente, en esta sección se definen las relaciones entre los distintos paquetes de vista.

2.2.5. IEEE 1471-2000.

Este modelo procura homogeneizar y ordenar la nomenclatura de descripción arquitectónica y homóloga los estilos como un modelo fundamental de representación conceptual, define un conjunto de recomendaciones centradas básicamente en 2 ideas, un marco conceptual para describir arquitecturas, y un conjunto de prácticas a seguir. La descripción de la arquitectura se

organiza en un conjunto de vistas, cada vista modela una parte del sistema y satisface uno o más intereses de las personas involucradas.

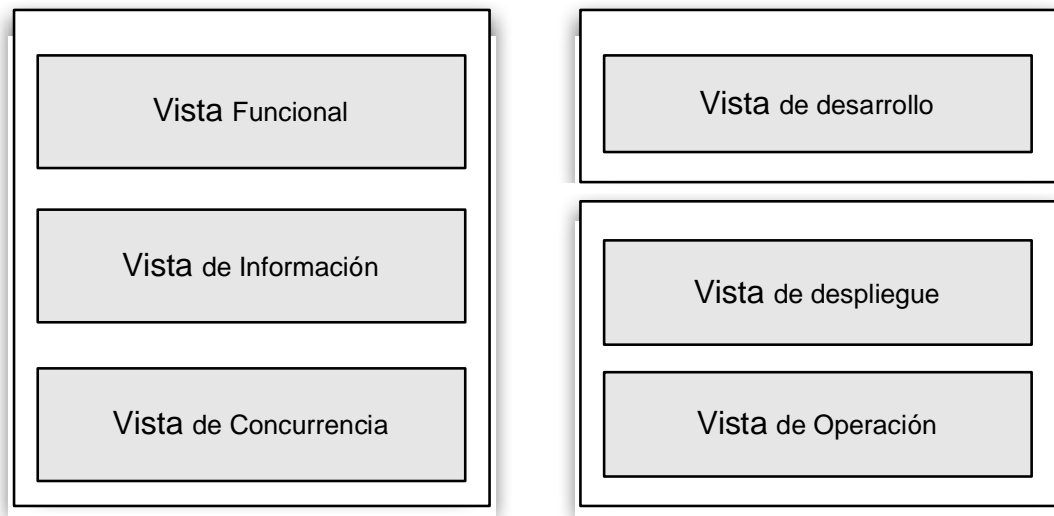


Figura 10 Agrupación de las vistas para el modelo 1471-2000

El objetivo de agrupar las vistas como se muestra en la figura 10 es debido a que la vista funcional, de información y concurrencia conforman la organización fundamental del sistema. La vista de desarrollo soporta la construcción del sistema y las vistas de despliegue y operacional son las vistas que soportarán al sistema una vez que este ya se encuentre operando.

2.2.5.1. Metas y objetivos IEEE1471.

- Definir la dirección para la incorporación de la arquitectura a los sistemas pensando en las normas de la IEEE.
- Establecer un marco conceptual y de vocabulario para hablar de problemas de arquitectura de los sistemas.

- Identificar y promulgar las buenas prácticas arquitectónicas.
- Permitir la evolución de las prácticas pertinentes a tecnologías maduras.

Si se desea que la documentación de la arquitectura cumpla con el estándar, se deben seguir las siguientes prácticas:

- **Identificación e información general.** En ésta práctica se lleva lo relacionado al control de versiones del documento, fecha, histórico de revisiones, estado, contexto del sistema, declaración del alcance, entre otros más.
- **Identificación del personal involucrado y sus intereses.** Se describen los diversos tipos de personas involucradas en el proyecto. Tales como, administradores, diseñadores, desarrolladores, usuarios, patrocinadores. A su vez, se incluye la diversidad de intereses que la arquitectura debe satisfacer.
- **Puntos de vista.** Cada de uno de éstos debe contener un nombre, personal involucrado, intereses que satisface, lenguaje o técnicas de modelado utilizados durante la construcción de la vista, algún método analítico para analizar de manera cualitativa o cuantitativa los atributos de calidad que satisface el punto de vista, y la justificación de éste.
- **Vistas.** Cada vista debe tener un identificador, una breve introducción y la representación del sistema con respecto a un punto de vista en particular.
- **Consistencia entre vistas.** En esta se registran las inconsistencias entre las vistas, así como algún tipo de procedimiento que indique la consistencia entre éstas.
- **Justificación.** Se debe incluir la justificación del tipo de arquitectura seleccionada, y de los puntos de vista utilizados.

2.2.5.2. Catálogo de Vistas propuesto por el IEEE1471.

A continuación se describe de forma detallada cada una de las vistas propuestas para este modelo:

2.2.5.2.1. Punto de vista Funcional.

Describe los elementos funcionales, sus responsabilidades, interfaces e interacciones primarias, esta dirige la forma de otras estructuras del sistema tales como la de información, de concurrencia y demás, el principal elemento a producir en esta vista es un modelo de la estructura funcional.

Los principales intereses que trata de dirigir esta vista son:

- **Capacidades funcionales.-** Definen lo que se requiere que realice el sistema de forma explícita o implícita así como lo que no debe de realizar.
- **Interfaces externas.-** Son los datos y flujos de control entre el sistema y otros, los datos pueden surgir de forma interna o externa y se necesita considerar la sintaxis y la semántica.
- **Estructuras internas.-** Las estructuras internas de un sistema son definidas por sus elementos internos, que es lo que hacen y cómo interactúan con otros, la organización interna puede tener un gran impacto en los atributos de calidad del sistema tales como disponibilidad, seguridad etc.
- **Diseño filosófico.-** El diseño filosófico se refiere a un conjunto de escenarios de calidad que se deben de listar con respecto a los intereses de cada *stakeholder*, por ejemplo, usuarios que sólo están interesados en la funcionalidad del sistema y las interfaces que

existen para los usuarios y otros sistemas, otro escenario podría ser el interés de un *stakeholder* en conocer como la arquitectura se adhiere a los principios de diseño o si el sistema es flexible y es fácil realizar cambios funcionales, de lo anterior se obtiene una colección de escenarios y se listan dentro de esta vista

La notación de esta vista puede estar representada por diferentes técnicas. El diagrama de componentes de UML es una buena elección ya que muestra los elementos del sistema, interfaces y conexiones entre los elementos. Existen otras notaciones como simples cajas y líneas, el beneficio de usarlas es que los *stakeholders* no técnicos particularmente los pertenecientes al negocio pueden tener un mejor entendimiento y aunque estos diagramas sean menos formales que un diagrama UML son muy útiles.

Se debe tener presente que en esta vista sólo se deben mostrar elementos funcionales, si se necesitan elementos para destacar otro tipo de elementos en esta vista se podrían estar sobrepasando sus límites.

Actividades

Para identificar los elementos funcionales [Clements01] propone las siguientes actividades:

- Trabajar con los requerimientos funcionales, tratando de obtener las responsabilidades clave.
- Identificar un conjunto de elementos funcionales que puedan ejecutar esas responsabilidades.
- Evaluar el conjunto identificado contra un criterio de diseño deseable

- Iterar los pasos anteriores hasta refinar la estructura que a juicio del arquitecto tenga sentido.

2.2.5.2.2. Punto de vista de Información.

Describe la forma en la que la arquitectura almacena, manipula, administra y distribuye la información, el principal producto de esta vista es la representación y documentación de las estructuras de datos.

Sus principales intereses en atender son:

- Información estructura y contenido
- Flujo de la información
- Pertenencia de los datos
- Calidad de datos
- Volumen de datos

El reto para la selección de la información y las estructuras es el enfocarse en los aspectos principales dejando los detalles a los modeladores y diseñadores de datos.

Se debe hacer un enfoque en un conjunto relativamente pequeño de entidades de datos y las relaciones entre ellos, decidir que entidades son las importantes dependiendo del problema que se está tratando de resolver y los intereses de los *stakeholders*.

El flujo de la información se refiere a la forma en la que la información fluye dentro del sistema y es accedida y modificada por sus elementos con lo cual se podría contestar a las siguientes preguntas:

- ¿Donde se crean y destruyen los datos?
- ¿Dónde son accedidos y modificados los datos?
- ¿Cómo es que los elementos individuales de información cambian y se mueven dentro del sistema?

La pertenencia de datos se refiere a la integración de los datos cuando estos pertenecen o se distribuyen en diferentes estructuras de datos asignados a diferentes sistemas las cuales se encuentran distribuidas físicamente en diferentes localidades.

En esta vista el modelado de datos es muy importante y se basa en tres tipos de modelos:

- Modelos de estructuras de datos estáticas, los cuales analizan la estructura de los datos.
- Modelos de información de flujo, los cuales analizan el movimiento dinámico de la información entre elementos del sistema.
- Modelos de información del ciclo de vida, Los cuales analizan la forma en la que los valores de los datos cambian con el tiempo.

Las estructuras de datos estáticas(figura 11) analizan la estructura estática de los datos, la importancia de los elementos de datos y su relación entre ellos. Un modelo de entidad-relación es una técnica de análisis de datos, la cual tiene una base matemática muy sólida, los elementos de interés son referidos como entidades, y sus partes que las constituyen como atributos. La semántica de los datos define relaciones estáticas entre las entidades.

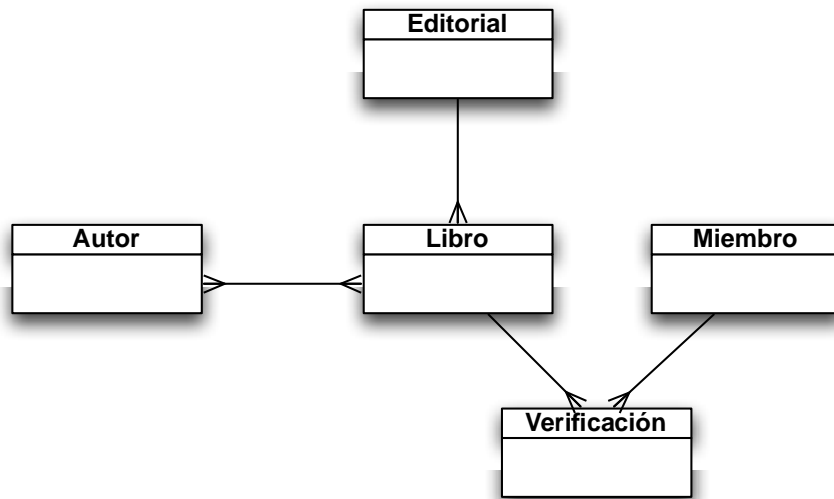


Figura 11. Ejemplo de un modelo de datos estático.

Los modelos de flujo de información analizan el movimiento dinámico de la información entre elementos del sistema con el mundo externo, estos modelos identifican los principales elementos arquitecturales y como la información fluye entre ellos. Cada flujo representa algunos datos que se transfieren de un componente a otro.

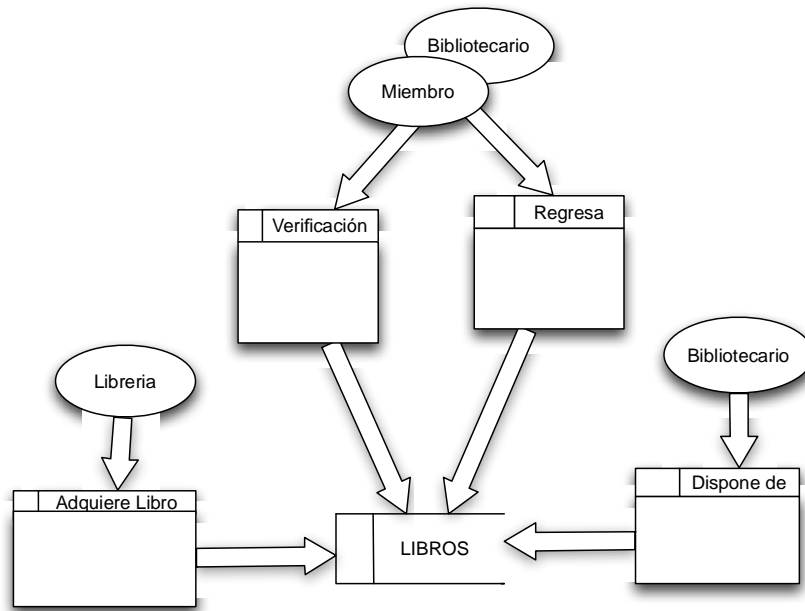


Figura 12. Diagrama de flujo de datos.

La notación del diagrama correspondiente a la figura 12 es la siguiente:

- Las flechas representan el flujo de datos.
- Las elipses representan entidades externas.

La información de la figura 12 se interpreta de la siguiente forma:

- Los miembros de la librería proveen de información para el proceso de verificación de la compra y el proceso de retorno.
- Una librería provee de información para el proceso de adquirir libros.

- El bibliotecario provee de información para disponer de los libros.
- Toda la información se escribe en el almacén de datos de los libros.

Para los modelos de ciclo de vida de la información se analiza la forma en que el valor de los datos cambia con el tiempo como se muestra en la figura 13.

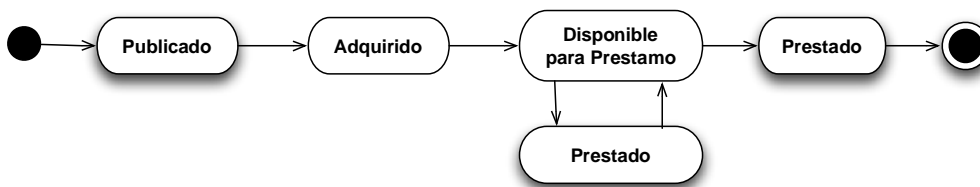


Figura 13. Diagrama de Ciclo de Vida.

Actividades

- **Proceso de normalización.-** Reduce el modelo a su forma más pura en la cual los datos no están repetidos ni son redundantes o existe información duplicada, por lo general los modelos no se llevan más allá de la tercera forma normal.
- **Análisis de dominio.-** Se realiza un análisis de los atributos o elementos y se definen sus valores permitidos.
- **Aplicar técnicas de descomposición o agregación.-** Se utiliza para derivar modelos de clases, la técnica de descomposición es utilizada para dividir elementos en piezas más pequeñas, mientras que la agregación es el proceso inverso crear nuevos elementos combinando otros.

2.2.5.2.3. Punto de vista de Concurrencia.

Describe cómo ciertas unidades funcionales pueden ejecutarse concurrentemente y cómo eso se coordina y controla los principales intereses de esta vista son:

- Estructura de tareas
- Mapeo funcional de elementos funcionales a tareas
- Comunicación entre procesos
- Sincronización e integridad

El aspecto más importante de crear una vista de concurrencia es el establecer la estructura de procesos del sistema, los cuales identifican la estrategia global para usar concurrencia en un sistema.

La vista de concurrencia mapea los elementos funcionales en tiempo de ejecución vía un modelo, este modelo típicamente contiene los siguientes elementos:

- **Procesos:** En este contexto el término proceso se refiere a un proceso del sistema operativo, esto es que direcciona un espacio que provee un entorno de ejecución para uno o más hilos independientes de ejecución. El proceso es la unidad básica de concurrencia en el diseño de un sistema.
- **Grupo de Procesos:** A nivel de arquitectura esto puede ser frecuentemente útil, ya que una colección de procesos puede ser considerada como una simple entidad a nivel sistema
- **Hilos.-** En este contexto el término hilo se refiere a un sistema operativo esto es un hilo de ejecución el cual puede independientemente calendarizar un proceso del sistema operativo.

- **Comunicación entre procesos:** Cuando los procesos están en ejecución, se asume que estos están aislados uno de otro, así que el proceso no puede cambiar nada de otro proceso, sin embargo en muchos sistemas concurrentes los procesos necesitan interactuar para poder coordinar su ejecución, peticiones de servicios entre ellos y paso de información, los procesos alcanzan esta interacción vía mecanismos de intercomunicación.

Actividades

Mapear los elementos a tareas.- En este proceso se necesita mapear todos los elementos funcionales a procesos, se deben de mapear estos elementos a procesos para mostrar que elementos se encuentran particionados entre diferentes procesos o que elementos corren en procesos compartidos. En este punto es muy importante conocer que la concurrencia solo se debe de utilizar cuando es necesaria ya que esta agrega complejidad al sistema y una significativa sobrecarga a la comunicación entre procesos.

- **Determinar el diseño de hilos.-** Este término se refiere al proceso de decisión en el número de hilos a incluir en cada procesos del sistema y como esos hilos son alojados y utilizados.
- **Considerar los recursos compartidos.-** Tan pronto como la concurrencia es introducida en el sistema, se debe de considerar como compartir los recursos entre los hilos concurrentes en ejecución.
- **Asignar prioridades a hilos y procesos.-** Se deben de considerar dentro del sistema que algunas tareas son más importantes que otras, si se tienen diferentes tareas de

diferente importancia ejecutándose en una máquina se necesita el control de la ejecución para que los procesos de mayor prioridad terminen antes que las de menor prioridad.

- **Analizar *Deadlocks*.**- Al introducir concurrencia en el sistema también se introduce el riesgo de que el sistema se pueda detener de muchas formas inesperadas como un *deadlock* o abrazo mortal, el cual se refiere a dos procesos los cuales compiten por recursos que pertenecen a cada uno y los cuales no se pueden liberar. Para evitar este tipo de errores se deben de revisar utilizando técnicas como redes de Petri las cuales permiten crear un modelo de procesos para detectar potenciales situaciones de *deadlocks*.
- **Analizar Contención.**- La contención ocurre entre tareas cuando más de una tarea requiere de recursos compartidos de forma concurrente y esto conlleva a que el sistema se vuelva demasiado lento. Para evitar esto se deben de analizar los recursos compartidos desde este punto de vista e identificar los posibles puntos de contención.

La notación utilizada para los modelos de estado típicamente es representada utilizando diagramas de transición de estados con UML.

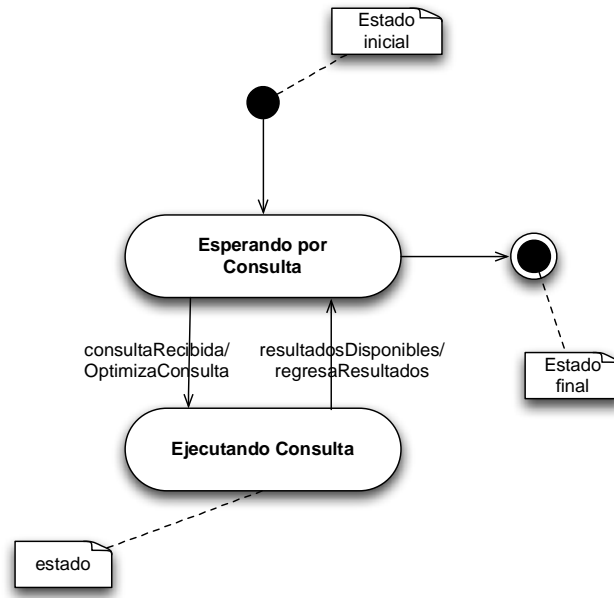


Figura 14. Ejemplo de diagrama de estados para un procesador de consultas.

2.2.5.2.4. Punto de vista de Desarrollo.

Describe los aspectos de interés para aquellos *stakeholders* involucrados en la construcción, pruebas, mantenimiento y mejora del sistema, los principales intereses de esta vista son:

- La organización de los módulos.
- Procesamiento común.
- Estandarización del diseño.
- Estandarización de las pruebas.
- Instrumentación.

La notación de este módulo es frecuentemente representada utilizando diagramas UML de componentes, se usa la representación de un paquete para representar un módulo de código y flechas para representar dependencias entre módulos. Si se requiere organización de nivel mas lato se puede utilizar paquetes y estereotipos.

Los productos a generar en esta capa son el modelo de la estructura de módulos, el modelo de diseño común y el modelo de codificación.

a) Modelo de la estructura de módulos.

Define la organización del código fuente del sistema en términos de módulos, una vez que se han identificado un conjunto de módulos en los cuales se pueden organizar los archivos fuentes, se puede utilizar una aproximación común para agrupar los módulos con abstracciones similares en capas. Estas capas se pueden organizar en una pila de dependencias en la cual se puede utilizar un criterio como de lo más abstracto o altamente funcional (en la cima de la pila) a lo funcional como se muestra en la figura 15.

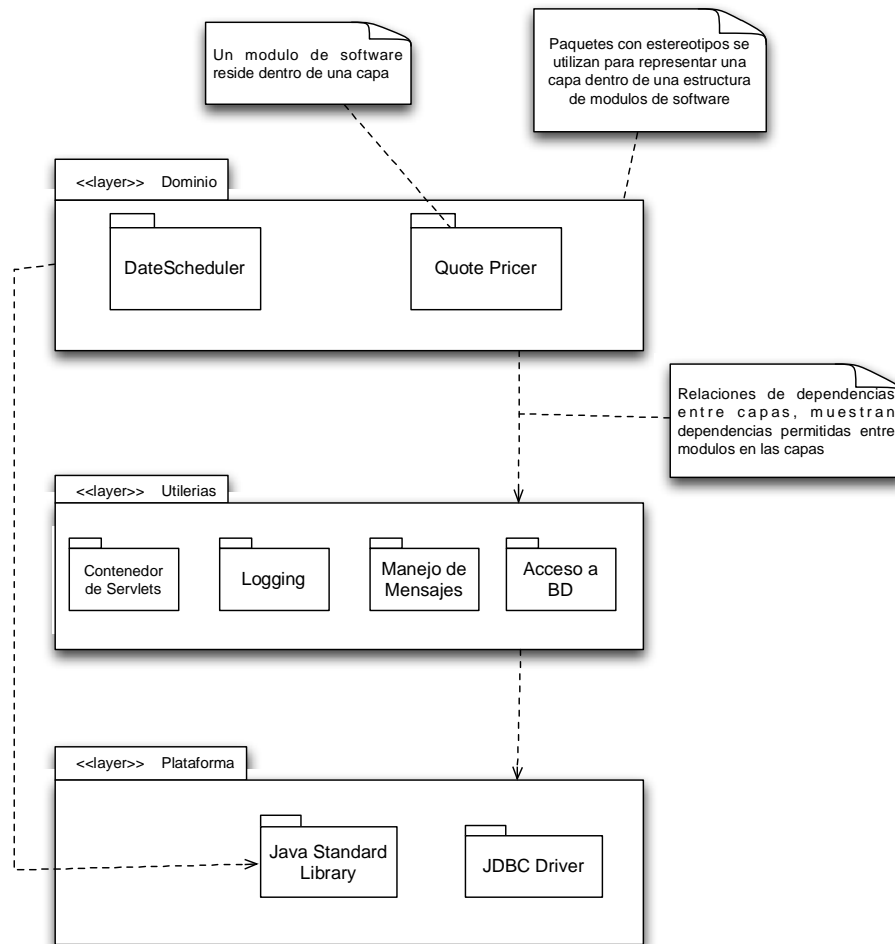


Figura 15 Módulos de un sistema.

Actividades

- **Identificar y clasificar los módulos.-** Agrupar el código fuente del sistema en un conjunto de módulos y clasificarlos.
- **Identificar las dependencias de los módulos.-** Identificar un conjunto claro de dependencias entre los módulos para que toda persona involucrada pueda tener un entendimiento claro del impacto de realizar
- **Identificar reglas de capas.-** Si se utiliza una aproximación basada en capas se debe diseñar un conjunto de reglas que determinan las llamadas que pueden realizar los

módulos de una capa hacia otra, es decir, si las llamadas en una capa sólo pueden realizarse dentro de la misma o si se pueden realizar hacia capas superiores o capas inferiores.

b) Modelos de diseño común.

Este modelo tiene tres importantes piezas:

- Una definición del procesamiento común, la cual requiere de elementos tales como:
 - Inicialización.
 - Terminación y reinicio de operaciones.
 - Mensajes, *logging* e instrumentación.
 - Internacionalización.
 - Uso de bibliotecas de terceros.
 - Procesos de configuración de parámetros.
 - Seguridad.
 - Manejo de transacciones.
 - Interacción con la base de datos.
- Una definición de los estándares de diseño a utilizar cuando se diseñen los elementos del sistema.
- Una definición del software común a utilizar y como este será utilizado.

Actividades.

Identificar procesos comunes.- Identificar que procesos comunes son requeridos, donde se requieren y como el procesamiento común debe de ser ejecutado.

- **Identificar restricciones de diseño.-** Identificar dentro de los procesos comunes si alguno de estos implica aspectos negativos al sistema y que puedan impactarlo de cierta forma.
- **Identificar y definir patrones de diseño.**
- **Definir el rol de los elementos estándar.-** Considerar si se tienen elementos estándar que puedan ser compartidos entre sistemas.

c) Modelos de Codificación.

El punto clave es definir la estructura del código, cómo es controlado, dónde diferentes tipos de código viven en esa estructura y cómo este debería de ser mantenido y extendido a través del tiempo. Un modelo de este tipo necesita capturar los siguientes elementos esenciales:

- Cómo el código debe de ser organizado dentro de archivos de fuentes.
- Cómo los archivos deben ser organizados en módulos.
- Qué estructura de directorios utilizar para mantener los archivos.
- Como el código debe ser construido para su liberación en forma de binarios.
- Qué tipo y alcance de pruebas se necesitan ejecutar de forma regular y cuando deben ser ejecutadas.
- Cómo el código fuente deberá ser liberado para pruebas y uso.
- Cómo el código deberá ser controlado utilizando la administración de la configuración.

Actividades

- Diseñar la estructura del código fuente.
- Definir una aproximación para la construcción.

- Definir el proceso de liberaciones.
- Definir la administración de la configuración.

2.2.5.2.5. Punto de vista de Despliegue.

Describe el ambiente en el cual el sistema será instalado incluyendo las dependencias que el sistema tiene en su propio entorno.

Los principales aspectos de interés de esta vista son:

- Tipos de hardware requeridos.
- Especificación y cantidad de hardware requerido.
- Requerimientos de software de terceros.
- Compatibilidad de las tecnologías.
- Requerimientos de red.
- Capacidad de red requerida.
- Restricciones físicas.

Los principales modelos a construir en esta vista son los modelos de plataforma de ejecución, modelos de red y modelos de dependencias de tecnología.

a) Modelos de plataforma de ejecución.

Este tipo de modelos son los principales de esta vista, es la descripción que define un conjunto de nodos de *hardware* que son requeridos, los cuales necesitan ser conectados con otros nodos vía una red u otras interfaces y alojan los elementos de *software*.

Un modelo de este tipo tiene los siguientes elementos principales:

- Nodos de procesamiento.

- Nodos clientes.
- Hardware de almacenamiento en línea.
- Hardware de almacenamiento fuera de línea.
- Ligas de red.

Los elementos anteriores se pueden visualizar dentro de la figura 16.

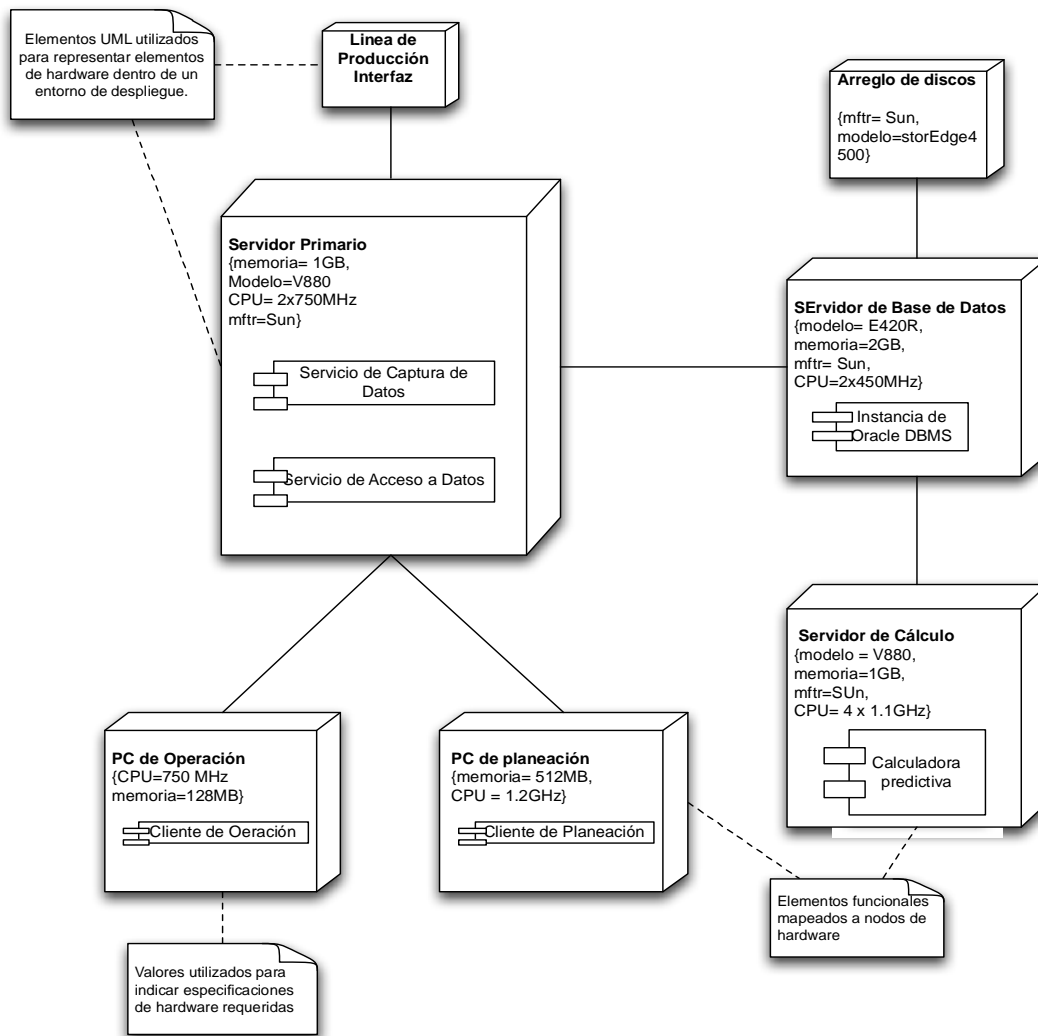


Figura 16.Ejemplo de Modelo de Plataforma.

b) Modelos de Red

Los elementos primarios de un modelo de red son:

- **Procesamiento de nodos.-** Representa los elementos del sistema que utilizan la red para transportar datos. Este conjunto de nodos puede coincidir con el conjunto de nodos del modelo de plataforma pero estas solo contienen información abstracta.
- **Nodos de Red.-** Indican el tipo de servicio de red que se espera esté disponible ya sea un *firewall* de seguridad, servicios de balanceo de cargas o de encriptación.

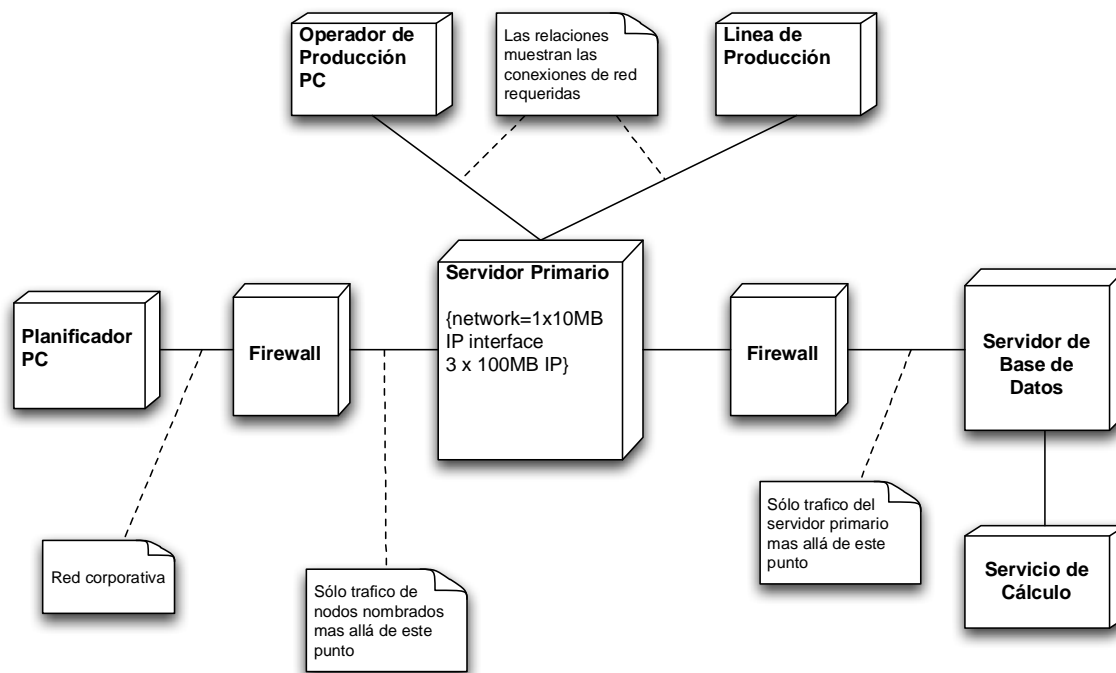


Figura 17. Ejemplo de Modelo de Red

Actividades.

- **Diseño de la Red.-** En esta etapa es necesario dibujar lo que se necesita de la red en términos de conexiones, capacidad y calidad de servicio. Todo esto representa lo que es efectivo de forma lógica más que de forma física.
- **Estimar la capacidad.-** Parte del diseño de la red lógica es el estimar la capacidad que se necesita entre los nodos. La precisión en esta etapa no es tan importante.

c) Modelos de dependencias de tecnología.

Las dependencias de tecnología son usualmente capturadas nodo por nodo de forma tabular, las dependencias de software son derivadas de la vista de despliegue donde se define el entorno utilizado por los desarrolladores.

Componente	Requerimientos
Servicio de acceso a datos	Solaris 8.0.2 Sun C++ 4.1.2 bibliotecas
Servicio de captura de datos	Solaris 8.0.2 Sun C++ 4.1.2 bibliotecas Oracle OCI bibliotecas 8.1.7.3
Sun C++ 4.1.2	Parche de Solaris 1534567 Parche de Solaris 1538367
Oracle OCI 8.1.7.3	Modulo opcional Solaris SUNWcpx Parche Solaris 1583956

Tabla 4. Dependencias de Software para el nodo del servidor primario de la figura.

2.2.5.2.6. Punto de vista de Operación.

Describe cómo el sistema será operado, administrado y soportado en el entorno de producción, sus principales intereses que atender son:

- Instalación y actualización.
- Migración funcional.
- Migración de datos.
- Monitoreo operacional y de control.
- Administración de la configuración.
- Soporte.

2.3. Elección de la Metodología

Actualmente es muy probable encontrar en la industria cualquiera de los enfoques anteriormente mencionados, ya que la forma de documentar una arquitectura ha evolucionado significativamente. Hoy en día la tendencia sobre esta práctica se centra en dos aspectos principales:

- Los arquitectos deben documentar las vistas que sean de mayor utilidad y no ajustarse a un número fijo de vistas.. Como es el caso en las propuestas de *Kruchten* y *Nord*.
- Documentar la arquitectura tomando en cuenta los intereses y necesidades de las personas involucradas en el proyecto. estos intereses se traducen como las cualidades que el sistema resultante debe poseer.

2.3.1. 4+1 Desventajas.

Es difícil encontrar una buena definición de los puntos de vista. El escrito original sólo provee ideas generales y es difícil encontrar definiciones más completas en fuentes accesibles, propiciando confusión cuando se quieren aplicar los puntos de vista.

Rational Unified Process ofrece más información en las vistas pero no se definen a nivel detallado. Cuando se habla de procesos tiende a entenderse como si se hablara de procesos de modelado de negocios cuando en realidad contiene información enfocada a performance y la disponibilidad, la vista lógica y física también se prestan a que existan diferentes interpretaciones entre los *stakeholders*, en general el conjunto de vistas no aborda explícitamente el direccionamiento de la información y los intereses del sistema.

Se puede decir que este conjunto de vistas no tiene una cobertura tan amplia ya que la falta de definiciones es un obstáculo inicial para nuestros clientes y nosotros mismos.

2.3.2. RM-ODP Desventajas.

Woods en su artículo [Rozanski05] menciona que no ha podido encontrar mucha evidencia de que este conjunto de vistas sea utilizado ampliamente por los profesionales en la industria.

Se asume que en las vistas de cómputo e ingeniería se está construyendo un sistema de objetos distribuido y se especifican primitivas que se deben de utilizar para describirlas.

En algunas vistas se asume que la propia notación de modelado puede ser usada para describirlas, lo cual no es ampliamente soportado por las herramientas.

2.3.3. Siemens Desventajas.

El enfoque de este conjunto de vistas está centrado más en sistemas de control más que en los sistemas de información como se muestra en la tabla 5.

4+1 (vistas)	RM ODP (vistas)	SIEMENS	SEI	IEEE 1471- 2000
Lógica	Empresa	Conceptual	Módulo	Funcional
Proceso	Información	Modulo	Conectores y Componentes	Información
Física	Computacional	Ejecución	Asignación	Concurrencia
Desarrollo	Ingeniería	Código		Desarrollo
Casos de Uso	Tecnología			Despliegue
				Operación

Tabla 5. Comparación de capas entre las diferentes propuestas analizadas.

Tomando en cuenta las recomendaciones anteriores, se utilizará para la documentación de la arquitectura de este trabajo el enfoque descrito por el IEEE 1471-200 ya que es el que se adecúa correctamente y es el que se encontró mayor detalle de la información que puede ser útil a todos los usuarios involucrados en el sistema a desarrollar.

CAPITULO III

Caso Práctico

DOCUMENTACIÓN DE LA ARQUITECTURA DEL SISTEMA DE CONCILIACIÓN DE POSICIONES Y MOVIMIENTOS DE INDEVAL

Este documento de Arquitectura de Software describe el diseño de alto nivel y arquitectura general del Sistema de Conciliación de posiciones y movimientos de Indeval (MIC), además explica los detalles de diseño a partir de diferentes vistas, donde cada vista es una descripción del sistema entero desde un punto de vista diferente.

3. Descripción General del Documento.

Esta sección describe el contenido y distribución general de este Documento de Arquitectura de Software.

Estructura del Documento.

Este documento contiene los siguientes capítulos:

Sección 3.1- Presenta el contenido del documento y explica cómo puede ser utilizado por cada uno de los *stakeholders* del proyecto.

Sección 3.2 – Describe la arquitectura a primer nivel. Es decir, las metas principales del diseño, los requerimientos de calidad centrales, cuáles fueron las decisiones más importantes y cuáles fueron las alternativas que se compararon.

Sección 3.3 – Detalla las decisiones tecnológicas que soportan la implementación de la arquitectura.

Sección 3.4 – Explica los principales requerimientos funcionales del sistema. Presenta los casos de uso principales que fueron tomados en cuenta para la elaboración de la arquitectura.

Sección 3.5 – Vista Funcional: Explica la estructura lógica de descomposición y estructura del sistema en subsistemas, componentes y servicios, los detalles de las clases principales del marco de trabajo de aplicación del sistema (aquellas que logran cumplir los requerimientos y servicios comunes de calidad).

Sección 3.6 – Vista de Información: Detalla las entidades principales derivadas del modelo de dominio del negocio.

Sección 3.7 – Vista de Desarrollo: Describe el ambiente de desarrollo que se utilizará para desarrollar la solución. Esta vista incluye también las justificaciones de las herramientas y

plataformas de desarrollo seleccionadas. Si se han decidido utilizar para algunas partes del sistema componentes de terceros en vez de desarrollarlos, aquí se describen los procesos de evaluación ejecutados y las justificaciones de los productos recomendados.

Sección 3.8 – Vista de Operación: Describe los mecanismos y decisiones que apoyan la operación y administración del sistema en el ambiente de producción. Se listan los mecanismos y procedimientos de instalación, monitoreo, administración, recuperación y diagnóstico del producto.

3.1.1. Guía de lectura para los Stakeholders.

Esta sección lista los roles principales de los involucrados en el sistema y cómo pueden usar este documento para comprobar que sus intereses han sido considerados en el diseño de arquitectura.

Integrante nuevo del proyecto: Leer el mapa de documentación (esta sección) para entender cómo se relacionan las diferentes vistas. Leer la descripción general del sistema (sección 3.3) para entender los aspectos de alto nivel del sistema y la justificación global del diseño. Por último, leer la sección 3.5 para entender la estructura lógica general del diseño de subsistemas y componentes.

Administrador del proyecto: Para soportar las tareas de planeación y administración del proyecto, concentrarse en el resumen de requerimientos (sección 3.4), de Información (sección 3.6) y desarrollo (sección 3.7).

Operaciones / Auditor de Seguridad: Leer la descripción general del sistema para un entendimiento de qué es lo que se necesita para instalar un ambiente funcional. Leer partes de la vista funcional y de información (secciones 3.4 a la 3.7) para estar familiarizado con los detalles de la estructura de la aplicación.

Cliente/Patrocinador: Leer la descripción general del sistema y la vista funcional (sección 3.5) para lograr un entendimiento general de la arquitectura y conceptos del sistema, para conocer cómo el sistema piensa resolver su misión y para ganar un entendimiento de cuál es el esfuerzo que se necesita para construirlo.

Usuarios: Los usuarios no estarán generalmente muy interesados en la documentación de la arquitectura del sistema. Sin embargo, pueden leer la primera parte de la vista funcional (sección 3.5) para tener un entendimiento general de cómo funciona el sistema.

Desarrolladores: Los programadores, *testers* y diseñadores son los principales lectores a los que el documento va dirigido. Prácticamente todas las vistas son relevantes para ellos y se recomienda que se comience con las descripciones generales (secciones 3.1 a la 3.5) y después se profundice poco a poco en cada una de las demás vistas en las secciones del sistema en donde le toca participar a cada quien.

3.1.2. Formato de documentación de Vista.

La arquitectura del sistema está descrita a partir de varias vistas relacionadas entre sí. Cada vista es un paquete de información acerca de ciertos aspectos específicos del sistema. Para entender completamente el diseño de la arquitectura se necesitan entender necesariamente todas las descripciones de cada una de las vistas utilizadas.

Esta sección describe la convención de documentación que se utiliza para cada una de las vistas:

1. Una **Presentación Inicial** muestra los elementos y relaciones que la vista se concentra en explicar, esta presentación inicial describe los elementos en el vocabulario de este tipo de vista. Normalmente, una vista describe sólo un subconjunto de todos los elementos existentes, con el objetivo de simplicidad y facilidad de entendimiento del diseño que la vista representa. La vista contendrá referencias a las convenciones de nomenclatura (por ejemplo, UML) y modelos (fórmulas, tipos de gráficos, estructura de tablas, etc.) que utiliza para representar los elementos que está explicando. Si se utiliza una nomenclatura informal (es decir, inventada por el arquitecto) se deberá incluir la simbología necesaria para que los lectores puedan entender el significado de las descripciones de diseño de la vista.

2. El **Catálogo de Elementos** arquitecturales de la vista, detallando al menos todos los elementos descritos en la presentación inicial. El catálogo normalmente incluye por lo menos:

- a. Alcance o descripción – El nombre de los tipos de elementos que se pueden utilizar en esa vista y su propósito o forma de uso.
- b. Tipos de Relación – Cada vista tiene varios tipos de relaciones que se pueden utilizar entre los elementos. Esta sección describe todos los tipos de relaciones que se incluyeron en la presentación inicial y otros tipos de relevancia secundaria omitidos por legibilidad.

3. **Antecedentes arquitecturales** que expliquen cómo se originó la propuesta de diseño presentada en la vista. El objetivo es justificar y convencer al lector de que la propuesta de diseño presentada es la más adecuada para el sistema. Estos antecedentes deben incluir:

- a. Justificación – Esta sección explica por qué fueron tomadas las decisiones de diseño reflejadas en la vista, cuáles eran las alternativas y por qué fueron estas rechazadas.
- b. Resultados del análisis – Esta sección documenta los resultados de análisis, pruebas ó investigaciones que hayan sido ejecutados para ayudar a determinar la estrategia de diseño presentada.
- c. Suposiciones – Esta sección documenta cualquier suposición hecha por el arquitecto ó equipo de arquitectura que pudiera afectar el diseño.
- d. Información adicional. – Cualquier información relevante para explicar la propuesta de diseño representada por la vista que no es contemplada en ninguna de las secciones anteriores.

3.2. Descripción General del Sistema.

Actualmente para el área de custodia de Indeval unos de los procesos que se necesita actualizar es el proceso de conciliación de posiciones y movimientos, este proceso consiste en verificar que el número de títulos que se encuentran resguardados por la institución y que pertenecen a otras bóvedas sea el correcto, en caso de ser contrario de deben de revisar todos los movimientos que se realizaron dentro de la bóveda por cada uno de los títulos para poder determinar la causa por la cual la conciliación no fue correcta.

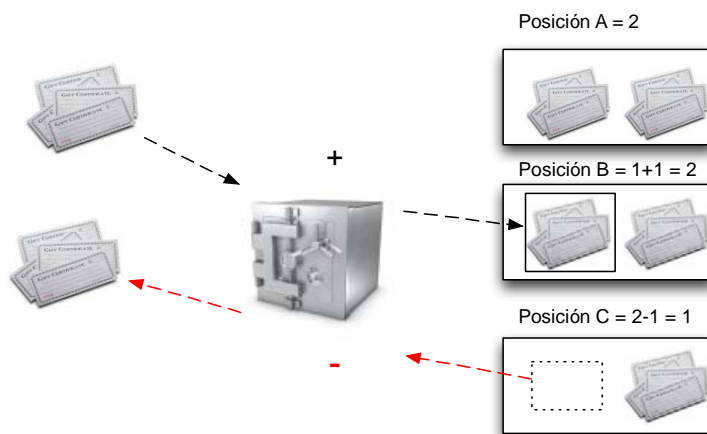


Figura 18. Movimientos de valores en una bóveda.

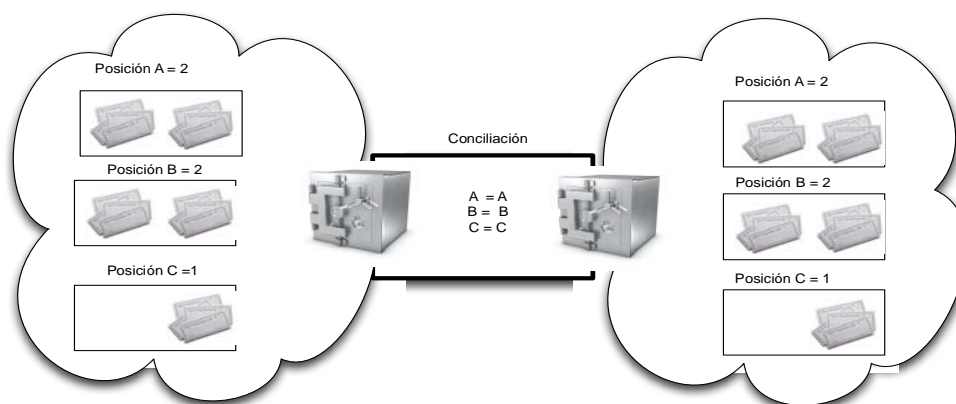


Figura 19 Proceso de Conciliación.

El proceso de conciliación consistirá en asegurar que los valores registrados en custodia a favor de nuestros depositantes tenga el respaldo de que existan físicamente depositados en las Bóvedas Físicas de Indeval instaladas en la Ciudad de México y en la Ciudad de Monterrey o, en su caso, en todos y cada uno de los custodios nacionales y extranjeros contratados por Indeval.

Al hacer mención de las bóvedas físicas de Indeval nos referimos a los registros de títulos contenidos en el Sistema de Control Numérico cuya contraparte será la cuenta de activos denominada “Bóveda Indeval”.

Al hacer mención de los custodios nacionales o extranjeros nos referimos a los estados de cuenta configurados con mensajes de PFI o *Swift* conocidos como MT535 y MT536 y archivos en formato TXT, donde las parejas comparables serán:

- Mensajes de PFI, MT535 y MT536 emitidos por BANCO DE MEXICO cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda Banxico”
- Mensajes tipo *Swift*, MT535 y MT536 emitidos por BANK OF NEW YORK cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda BONY”
- Mensajes tipo *Swift*, MT535 y MT536 emitidos por Euroclear cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda EUROCLEAR”
- Mensajes tipo *Swift*, MT535 y MT536 emitidos por CLEARSTREAM cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda CLEARSTREAM”
- Mensajes tipo *Swift*, MT535 y MT536 emitidos por BANCO SANTANDER ESPAÑA cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda SANTANDER”
- Mensajes tipo *Swift*, MT535 y MT536 emitidos por BANCO BBVA ESPAÑA cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda BBVA”
- Mensajes tipo *Swift*, MT535 y MT536 emitidos por JP MORGAN cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda JP MORGAN”

- Archivo tipo TXT, emitido por DEUTSCHE BANK cuya contraparte de comparación será la Cuenta de Activos denominada “Bóveda DEUTSCHE”

3.2.1. Contexto del Sistema.

El módulo de conciliación es un módulo usado para dar servicios de forma interna al área de custodia de Indeval, la figura 18 muestra el diagrama de contexto del módulo de conciliación y las relaciones existentes de los componentes internos y externos.

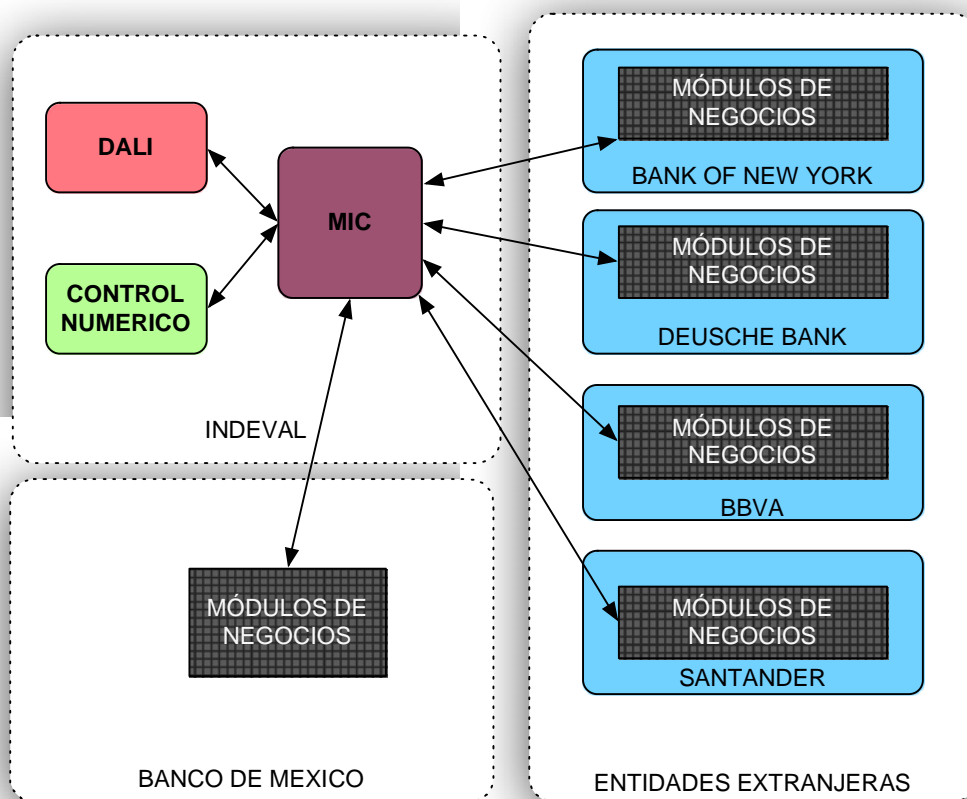


Figura 20. Diagrama de Contexto MIC.

3.2.2. Principios Arquitecturales.

- El sistema de conciliación realizará la conciliación de las bóvedas al final del día cuando la operación de Dalí haya terminado. Si se requiere realizar conciliaciones intradía éstas se realizarán con la información que exista en ese momento en las bóvedas.
- La conciliación con bóvedas extranjeras se realizará con información del día anterior, ya que por motivos de husos horarios diferentes la información con los custodios extranjeros llega desfasada en horarios.
- Se necesita que un operador sea el que realice el análisis de las posiciones con diferencias y decida qué área es la que atenderá el incidente así como la prioridad del mismo.
- La información obtenida de las bóvedas externas se debe almacenar en una estructura homogénea dentro del Mic para que diferentes fuentes de datos que cumplan con la estructura elegida puedan ser conciliadas.

3.2.3. Atributos de Calidad.

Los atributos de calidad forman parte de los requerimientos no funcionales del sistema. Son características medibles que permiten expresar y evaluar el grado de satisfacción de los usuarios y/o diseñadores (es decir la calidad) con respecto al sistema.

Rendimiento

El rendimiento se refiere al tiempo que requiere el sistema para responder a los eventos que lo afectan en un intervalo de tiempo.

Para el sistema de conciliación se requiere que las comparaciones que se realizan entre las posiciones de las diferentes bóvedas y sus movimientos sean sumamente rápidas como para procesar 2000 operaciones por minuto ya que es un tiempo razonable para que los operadores del sistema puedan esperar.

Usabilidad

Se refiere a la facilidad que provee el sistema a los usuarios para completar las tareas deseadas y el tipo de usuarios que proveen su soporte.

Para el sistema de conciliación se requiere que se pueda acceder a todos los servicios del sistema mediante una interfaz grafica, se pueden utilizar las siguientes pantallas para agrupar todos los servicios:

- Pantalla de conciliación.- Donde se muestren las comparaciones de las posiciones que se realizan de las diferentes bóvedas.
- Pantalla para administración de notificaciones.- Donde se muestren y envíen las notificaciones de los incidentes ocurridos dentro del sistema.
- Pantalla para administración general del sistema.- Donde se puedan modificar los siguientes catálogos del sistema:
 - Usuarios
 - Parámetros
 - Áreas

- Grupos

Seguridad

Es una medida del sistema para resistir el uso del mismo sin autorización mientras éste provee sus servicios a usuarios legítimos.

Dentro del sistema de conciliación se necesita un esquema basado en usuario y *password* para poder acceder a la aplicación, además se necesita que se pueda asignar un perfil a cada uno de los usuarios para que se puedan asignar diferentes permisos sobre las operaciones del sistema.

Las aplicaciones actuales de Indeval ya cuentan con un esquema de seguridad, el cual está basado en los servicios de seguridad proporcionados por el marco de trabajo de *AquaLogic Enterprise Security* (ALES). Este marco de trabajo es una solución enfocada hacia un control de acceso muy granular sobre los recursos que protege y que funciona de forma centralizada de manera tal que proporciona una infraestructura de seguridad unificada y adaptable para el aseguramiento de aplicaciones distribuidas; el mecanismo de autenticación utilizado para el desarrollo de la aplicación de conciliación será mediante la tupla clave de usuario, contraseña, sistema para acceder a la aplicación a través de un navegador.

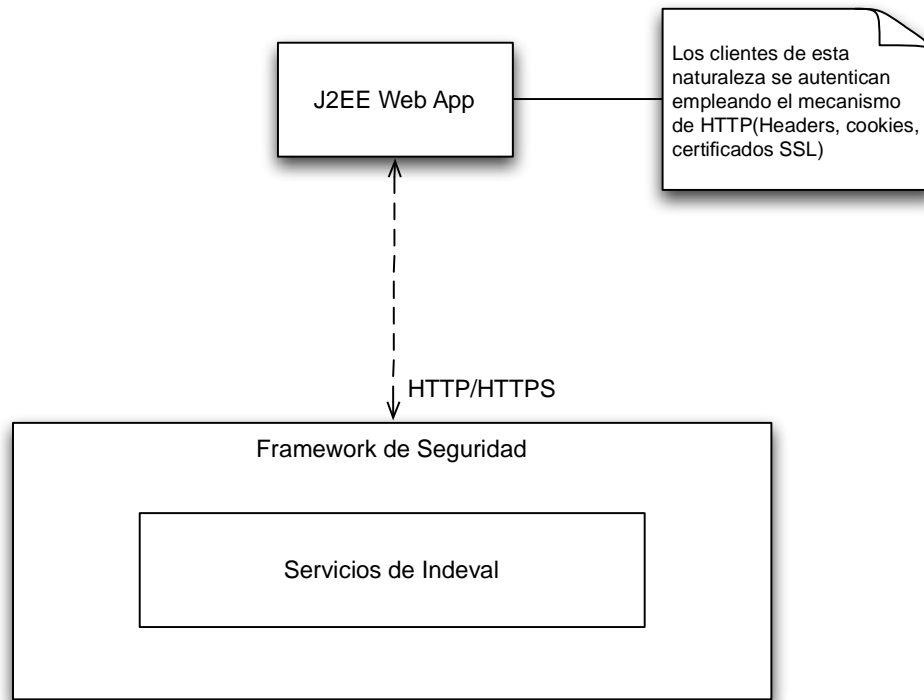


Figura 21. Modelo de seguridad para el sistema de conciliación.

3.3. Mapa de Tecnologías.

En esta sección se explicará las características principales de cada una de las tecnologías mas importantes a utilizar en la construcción del sistema.

3.3.1. Herramientas de Soporte.

Se requiere de una herramienta para que el proceso de compilación del código fuente, la creación de bibliotecas y productos para su instalación sea un proceso automatizado. La herramienta a utilizar es Maven 2.

Las características de esta herramienta son:

- Administración de dependencias.
- Reduce el tamaño del proyecto ya que las bibliotecas y marcos de trabajo utilizados son extraídos de un repositorio central.
- Administración del ciclo de vida del proceso de construcción y ensamblado.

3.3.2. Herramientas para Pruebas

Las pruebas de unidad son pequeñas porciones de código destinadas a probar un componente o clase de manera individual.

Las pruebas de unidad tienen las siguientes ventajas:

- Facilitar el rastreo, control e implementación de cambios.
- Facilitar la integración entre componentes.
- Servir de documentación para conocer el uso y comportamiento del componente.
- Separar la implementación de tecnologías alrededor del componente.

Las pruebas de unidad serán implementadas con *Junit*, este producto está integrado ya con la mayoría de los productos comerciales, lo cual provee una justificación para su uso y confiabilidad.

3.3.3. Herramientas de Integración.

El interés en desarrollar aplicaciones cada vez más ligeras y con mejor desempeño ha llevado a la construcción de herramientas denominadas contenedores ligeros, estos contenedores habilitan la total separación de código de ensamblaje entre componentes y código de plomería

(apertura de conexiones, cierre de sesiones, búsqueda de propiedades, etc), permitiendo que el desarrollador sólo escriba código que satisfaga los requerimientos del negocio. Dicho código de plomería se define en un archivo XML de configuración.

Cabe señalar que dichos contenedores no suplen de ninguna manera los contenedores de los servidores de aplicación, y que sus funciones son diferentes. Si se complementan éstos se pueden atender requerimientos de robustez, escalabilidad, y rendimiento.

Spring Framework es un marco de trabajo y la alternativa que se eligió para usar dentro del MIC debido a las siguientes razones:

- Provee una manera centralizada y automatizada de configurar objetos y componentes de la aplicación.
- No es invasivo debido a que no existe código del marco de trabajo mezclado con el código de la aplicación.
- Facilita la programación orientada a aspectos.
- Integración con tecnologías de presentación.
-

3.3.4. Motor de Reportes.

El motor de reportes será *Jasper Reports*, este marco de trabajo está dirigido a desarrolladores y sus características más importantes son:

- Permite que los reportes incluyan varias columnas e imágenes.
- Permite la extracción de datos directamente de la base de datos utilizando JDBC.

- Los reportes pueden salir a la pantalla, impresora o archivos con formatos PDF,HTML,XLS,CSV y XML.
- Anidación de reportes dentro del reporte principal.
-

3.3.5. Herramientas de desarrollo para la capa de presentación.

Flex

Flex fue inicialmente liberado como una aplicación de la J2EE o biblioteca de etiquetas JSP que compilaba el lenguaje de marcas Flex (MXML) y ejecutaba mediante *ActionScript* aplicaciones Flash (archivos SWF binarios). Versiones posteriores de Flex soportan la creación de archivos estáticos que son compilados, y que pueden ser distribuidos en línea sin la necesidad de tener una licencia de servidor.

El objetivo de Flex es permitir a los desarrolladores de aplicaciones web construir rápida y fácilmente Aplicaciones de Internet Ricas, también llamadas RIAs. En un modelo multi-capa, las aplicaciones Flex son el nivel de presentación.

Flex pone en relieve el desarrollo de Interfaces gráficas de usuario usando un lenguaje XML llamado MXML. Flex tiene varios componentes y características que aportan funcionalidades tales como Servicios Web, objetos remotos, arrastrar y soltar, columnas ordenables, gráficas, efectos de animación y otras interacciones simples. El cliente sólo carga la aplicación una vez, mejorando así el flujo de datos frente a aplicaciones basadas en HTML (PHP, ASP, JSP, CFMX), las cuales requieren de ejecutar plantillas en el servidor para cada acción. El lenguaje y la estructura de archivos de Flex buscan el desacoplamiento de la lógica y el diseño.

Las ventajas de Flex sobre otro tipo de aplicaciones web son:

- Es fácil de implementar flujos de operaciones que involucran más de un paso, ya que el cliente mantiene información del contexto.
- El cliente puede realizar cierto procesamiento: validación, formateo, filtrado, ordenamiento, búsquedas, etc.

RobotLegs

Robotlegs es una microarquitectura escrita completamente en AS3 (*Action script*) para el desarrollo de aplicaciones Flex, Flash y Air, cuyo principal objetivo es interconectar las diferentes partes de la aplicación y proveer un mecanismo para que se comuniquen, en este caso utiliza el sistema de mensajería nativo de Flex.

El marco de trabajo viene con una implementación de referencia basada en el patrón MVC+S que sirve de guía para estructurar la aplicación, que incluye las clases *Mediator*, *Command* y *Actor* (denominadas los actores del marco de trabajo), y aunque no es obligatorio su uso constituye un punto de comienzo. La inyección de dependencias la realiza a través de la biblioteca *SwiftSuspenders*, aunque deja abierta la posibilidad de usar otras, los puntos de inyección de dependencias se indican declarativamente a través de metadatos.

3.3.6. Servidor de Aplicaciones.

El entorno de negocio de hoy en día demanda aplicaciones Web y de comercio electrónico que aceleren nuestra entrada en nuevos mercados, nos ayude a encontrar nuevas formas de llegar y de retener clientes, y nos permita presentar rápidamente productos y servicios. Para construir y desplegar estas nuevas soluciones, necesitamos una plataforma de comercio electrónico probada y creíble que pueda conectar y potenciar a todos los tipos de usuario mientras integra nuestros datos corporativos, las aplicaciones mainframe, y otras aplicaciones empresariales en una solución de comercio electrónico fin-a-fin poderosa y flexible. Nuestra solución debe proporcionar el rendimiento, la escalabilidad, y la alta disponibilidad necesaria para manejar nuestros cálculos de empresa más críticos.

Como una plataforma líder en la industria de comercio electrónico, WebLogic Server nos permite desarrollar y desplegar rápidamente, aplicaciones fiables, seguras, escalables y manejables. Maneja los detalles a nivel del sistema para que podamos concentrarnos en la lógica de negocio y la presentación.

WebLogic Server es un servidor de aplicaciones con las siguientes características:

- Es una plataforma para aplicaciones empresariales multi-capa distribuidas
- Centraliza los servicios de aplicación como funciones de servidor web, componentes del negocio, y acceso a los sistemas "*backend*" de la empresa
- Utiliza tecnologías como el almacenamiento en memoria inmediata y almacenes de conexiones para mejorar la utilización de recursos y el funcionamiento de la aplicación.

- Proporciona facilidades a nivel de seguridad empresarial y una administración poderosa, funciona en la capa media (o capa "n") de una arquitectura multi-capa,
- Implementa J2EE, el estándar empresarial de Java.

3.3.7. Herramientas para la calendarización de tareas.

Quartz

Quartz es una biblioteca Java para programar la ejecución de tareas. Permite "agendar" trabajos a realizar, con una sintaxis similar a la del cron de Unix, puede correr en prácticamente cualquier entorno Java, ya sea una Aplicación Java SE o una Aplicación Java EE además, es una biblioteca para poder agendar tareas y programar su ejecución en momentos específicos.

De manera realmente muy simple, es posible programar una tarea para que se ejecute:

- cada una cantidad de tiempo fija (ej: cada 1 hora)
- repetidamente, en momentos determinados (ej: a las 10 de la noche de los martes y jueves)
- en rangos de tiempo (del 10 al 15 de cada mes, a las 9 de la mañana)
- en saltos de tiempo (los días impares del mes, a las 10 de la noche)

Quartz consiste muy básicamente de 3 componentes:

1. La tarea a ejecutar (que implementa la interfaz Job)

2. El *scheduler* (que administra y ejecuta las tareas)
3. El *trigger* (el evento que activa en el *scheduler* la ejecución de la tarea)

Spring provee soporte para crear tareas en *Quartz* de forma declarativa. Así, a través de la declaración de *beans*, podremos ejecutar periódicamente métodos de cualquier de los *beans* declarados en Spring.

3.3.8. Lenguajes de Programación.

Java es toda una tecnología orientada al desarrollo de software con el cual podemos realizar cualquier tipo de programa. Hoy en día, la tecnología Java ha cobrado mucha importancia en el ámbito de Internet gracias a su plataforma J2EE.

La tecnología Java está compuesta básicamente por 2 elementos: el lenguaje Java y su plataforma. Con plataforma nos referimos a la máquina virtual de Java (Java Virtual Machine).

Una de las principales características que favoreció el crecimiento y difusión del lenguaje Java es su capacidad de que el código funcione sobre cualquier plataforma de software y hardware. Además es un lenguaje orientado a objetos que resuelve los problemas en la complejidad de los sistemas, entre otros.

3.4. Requerimientos Funcionales.

Aunque la lista de requerimientos funcionales no es parte de la descripción de arquitectura, es útil entender los principales escenarios y casos de uso como marco para el diseño de la estructura funcional del sistema.

RQ1 Conciliación de posiciones de múltiples bóvedas contra Dalí.

Tipo	Funcional
Prioridad	Alta
Descripción	El sistema debe de poder realizar la conciliación de múltiples bóvedas, por lo cual deberá tener una estructura de datos homogénea que permita incluir diferentes fuentes de información.
Autor	Gerente de la Bóveda
Ascendencia	

RQ2 Configuración del sistema para conciliaciones automáticas y manuales.

Tipo	Funcional
Prioridad	Alta
Descripción	El sistema deberá de proveer la funcionalidad de configurar un horario de ejecución para el proceso de conciliación, además deberá de tener la posibilidad de que el administrador del sistema las pueda ejecutar en cualquier momento
Autor	Gerente de la Bóveda.
Ascendencia	

RQ3 Sistema de notificaciones de errores para las conciliaciones

Tipo	Funcional
Prioridad	Alta
Descripción	<p>El sistema deberá de proveer un sistema de notificación basado en usuarios, grupos y áreas, se podrán dar de alta múltiples áreas a las cuales podrán pertenecer grupos, los grupos no podrán ser más de 4 por área y a los grupos se unirán los usuarios del sistema.</p> <p>El sistema deberá de relacionar cada uno de los errores del sistema a una notificación, la notificación se relacionará a un área a la cual le llegará por medio de los usuarios pertenecientes.</p> <p>Las notificaciones llegarán a cada uno de los usuarios del sistema dependiendo del nivel del grupo, los niveles dentro de los grupos va del 1 al 4 donde el 1 tiene menor prioridad y 4 la mayor.</p> <p>Las notificaciones se enviarán de acuerdo a un parámetro configurado en el sistema en el cual se podrá configurar cada cuantos días se debe de revisar el estatus de las notificaciones.</p> <p>Las notificaciones deberán de tener una propiedad que indique la prioridad de la notificación, esta notificación deberá de cambiar de acuerdo a las veces que se ha notificado a los usuarios, las notificaciones se envían a todos los grupos de acuerdo a la prioridad de la notificación y a los niveles de los grupos dentro del área.</p>

	<p>Notificación nivel 1. Llega a los usuarios del área correspondientes al grupo de nivel más bajo.</p> <p>Notificación de nivel 2 Llega a los usuarios del área correspondientes al grupo de nivel más bajo y al grupo superior.</p> <p>Notificación de nivel 3 Llega a los usuarios del área correspondiente al grupo de nivel más bajo y a los dos grupos superiores</p> <p>Notificación de nivel 4 Llega a todos los usuarios del sistema de todos los grupos</p>
Autor	Gerente de la Bóveda
Ascendencia	

Cierre automático de partidas

Tipo	Funcional
Prioridad	Media
Descripción	Automáticamente cuando todas las notificaciones que pertenecen a una partida se cerraron, la partida se debe de cerrar de forma automática.
Autor	Gerente de la Bóveda
Ascendencia	

Prioridad Media

Descripción	Las pantallas del sistema deben de ser fáciles de operar, se requiere que para cada acción del sistema no lleve más de tres clics en realizarlas
Autor	Gerente de la Bóveda
Ascendencia	

3.4.1. Casos de Uso del Sistema.

En la figura 22 se presentan los casos de uso centrales del sistema:

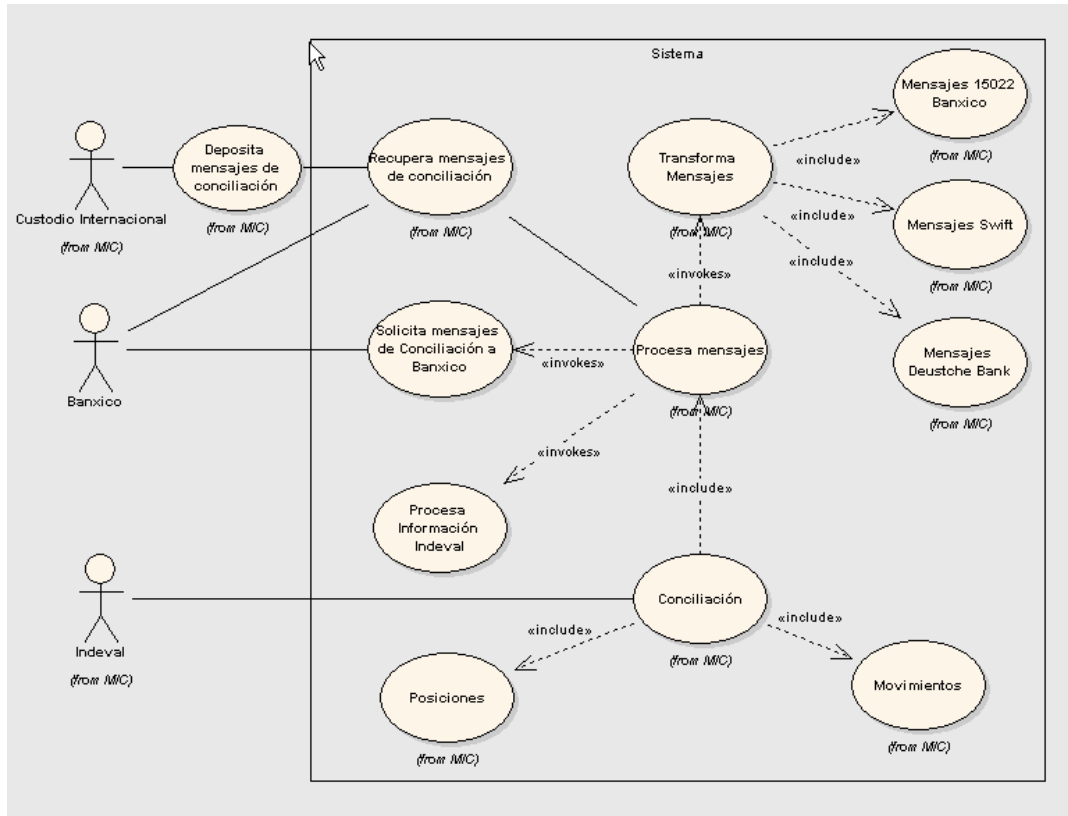


Figura 22 Casos de uso del Sistema.

3.5. Vista Funcional

3.5.1. Estructura de Subsistemas.

La figura 23 muestra la estructura propuesta para la construcción del sistema y los módulos que la conforman, las líneas muestran la relación directa que existe entre ellos.

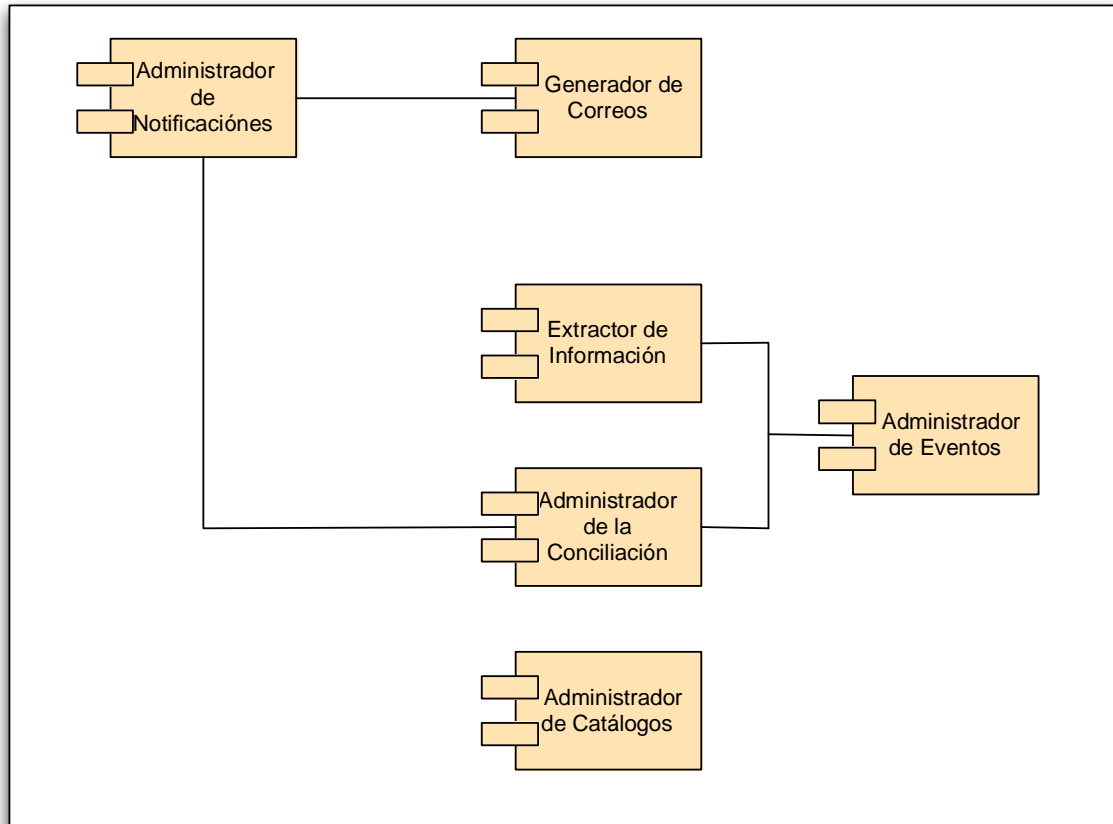


Figura 23 Principales componentes del Módulo de Conciliación.

3.5.1.1. Catálogo de Elementos.

3.5.1.1.1. Extractor de Información.

El componente de extracción, contiene los servicios de extracción de datos de los sistemas que proveen las posiciones para la conciliación, las responsabilidades de este componente son:

- Ocultar la implementación de la extracción de datos de fuentes externas.
- Controlar y sincronizar la ejecución de los procesos de extracción de datos.

- Transformar las estructuras de datos de los diferentes proveedores de información a una estructura determinada por el sistema.

3.5.1.1.2. Administrador de la Conciliación.

El administrador de la conciliación, es el componente que contiene los servicios dedicados a la comparación de posiciones y movimientos, las responsabilidades de este componente son:

- Realizar la comparación tanto de posiciones y movimientos de las diferentes bóvedas.
- Controlar y sincronizar la ejecución de los procesos de verificación de partidas abiertas.
- Proveer servicios para la extracción de posiciones y movimientos del Sistema.

3.5.1.1.3. Administrador de Notificaciones.

El administrador de notificaciones, es el componente que contiene los servicios dedicados al envío de notificaciones vía correo electrónico dentro del sistema, contiene internamente un algoritmo el cuál decide como enviar las notificaciones de acuerdo a dos criterios, el área a la que va dirigida la notificación y su importancia. Las responsabilidades de este componente son:

- Proveer los servicios para la creación de nuevas notificaciones a usuarios del sistema.
- Controlar y sincronizar los procesos automáticos de reenvío de notificaciones abiertas de acuerdo a los parámetros establecidos dentro del sistema.

- Controlar y sincronizar los procesos automáticos de cierre de notificaciones.
- Decidir los destinatarios a los que va dirigida una notificación de acuerdo a su importancia y el tiempo transcurrido desde su creación para el envío de los correos electrónicos.

3.5.1.1.4. Generador de Correos.

El generador de correos, es un componente que sirve como herramienta de generación y envío de correos electrónicos. Esta herramienta es usada por el componente que administra las notificaciones y sus responsabilidades son:

- Crear por medio de plantillas preestablecidos el cuerpo de los mensajes de correo electrónico que envía el sistema.
- Administrar las plantillas de correo electrónico que se usan en el sistema.
- Enviar los mensajes de correo a los destinatarios recibidos.

3.5.1.1.5. Administrador de Eventos.

El administrador de eventos es un componente que provee servicios internos a los componentes para el registro de eventos dentro del sistema, inicialmente sólo presta estos servicios a los componentes de conciliación y extracción de datos, este componente registra los eventos de extracción de datos y de ejecución de conciliaciones para posteriormente poder evaluar la información, las responsabilidades de este componente son:

- Generar un registro de los eventos internos del sistema indicando la fecha y hora de la ejecución del evento, el nombre del evento que se ejecuto, el estado de la ejecución del evento y una descripción de error generada por el sistema en caso de que el estado resultante haya sido de error.

3.5.1.1.6. Administrador de Catálogos.

El administrador de Catálogos es el componente que administra los catálogos del sistema y provee servicios para agregar, modificar y borrar los elementos de cada uno de los catálogos del sistema.

3.5.1.2. Framework de Aplicación.

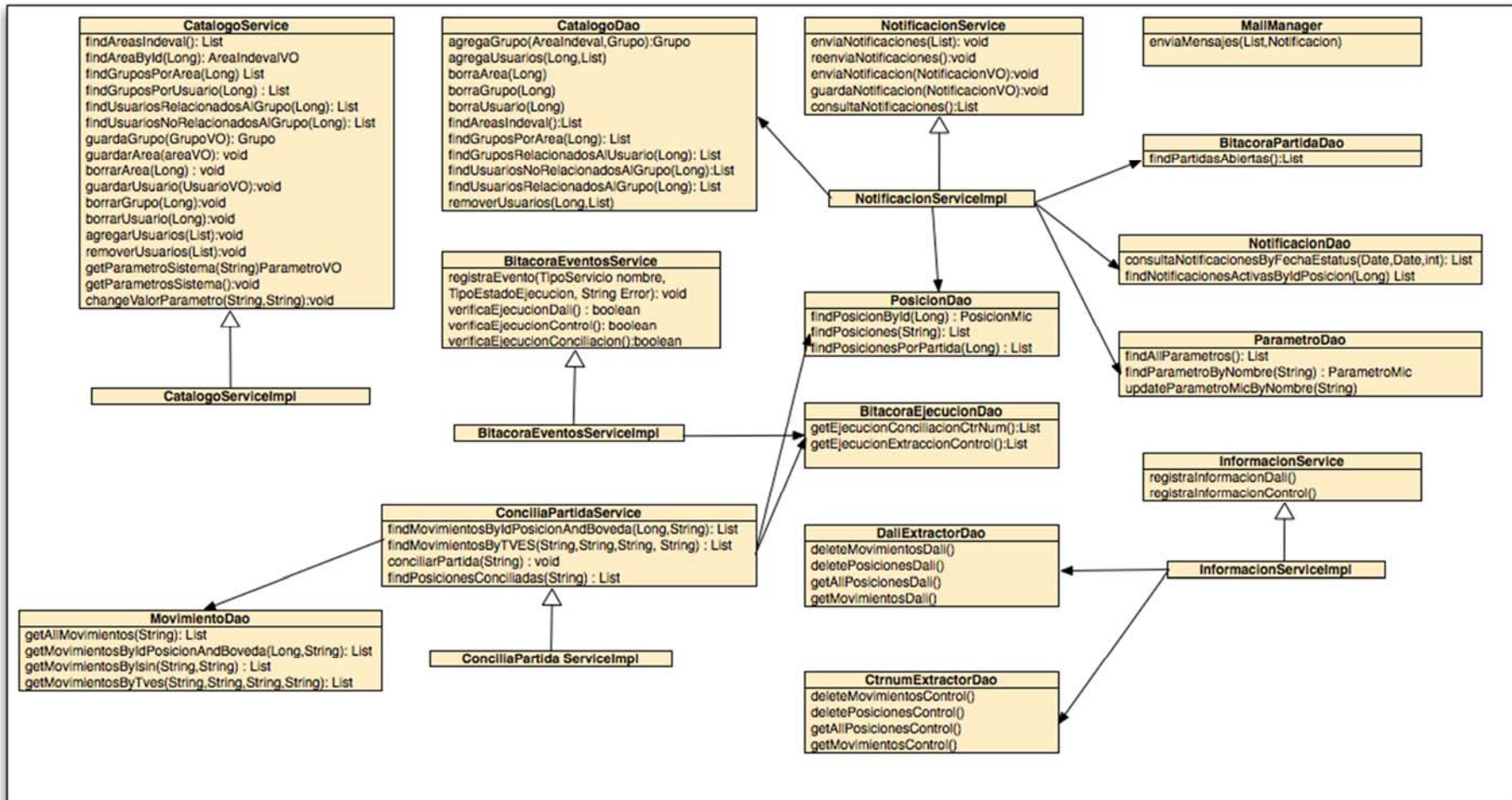


Figura 24. Elementos principales del sistema MIC.

3.5.1.2.1. CatalogoService.

Contiene los servicios que se utilizan para la administración de los catálogos del sistema

(Usuarios, Grupos, Áreas y Parámetros), sus principales métodos se muestran en la tabla 6:

Regresa una lista de todas las áreas de Indeval:
• <i>findAreasIndeval(): List</i>
Regresa un área específica correspondiente al id:
• <i>findAreaById(Long): AreaIndevalVO</i>
Busca todos los grupos y devuelve una lista de todos los grupos que están relacionados al área correspondiente al id:
• <i>findGruposPorArea(Long) List</i>
Busca todos los grupos y devuelve una lista de todos los grupos a los que esta relacionado un usuario determinado por su id:
• <i>findGruposPorUsuario(Long) : List</i>
Devuelve una lista de usuarios relacionados al grupo de acuerdo a su identificador:
• <i>findUsuariosRelacionadosAlGrupo(Long): List</i>
Devuelve una lista de usuarios que no están relacionados a un grupo determinado por su identificador:
• <i>indUsuariosNoRelacionadosAlGrupo(Long): List</i>
Persiste el objeto GrupoVO y devuelve el objeto
• <i>guardaGrupo(GrupoVO): Grupo</i>
Persiste el objeto AreaVO que representa un area de Indeval
• <i>guardarArea(areaVO): void</i>
Borra un area determinada por su identificador
• <i>borrarArea(Long) : void</i>
Persiste el objeto UsuarioVO que representa un usuario del sistema
• <i>guardarUsuario(UsuarioVO):void</i>
Borra un grupo determinado por su identificador
• <i>borrarGrupo(Long):void</i>
Borra un usuario determinado por su identificador
• <i>borrarUsuario(Long):void</i>
Relaciona una lista de usuarios a un grupo
• <i>agregarUsuarios(List):void</i>
Elimina una relacion de usuarios de un grupo
• <i>removeUsuarios(List):void</i>
Obtiene un parámetro del sistema por su nombre
• <i>getParametroSistema(String)ParametroVO</i>
Obtiene una lista de todos los parametros del sistema
• <i>getParametrosSistema():List</i>
Cambia el valor de un parámetro determinado por el nombre.
• <i>changeValorParametro(String,String):void</i>

Tabla 6. Principales Métodos del servicio de Catálogos.

3.5.1.2.2. BitacoraEventosService.

Registra en base de datos la ejecución de los eventos determinados por el tipo de servicio y el estado en el que terminaron después de su ejecución. Además provee consultas del estado de algunos servicios como la verificación del estado de las conciliaciones ejecutadas, sus principales métodos se muestran en la tabla 7:

Persiste en la base de datos un servicio ejecutado dentro del sistema y el estado en el cual terminó este servicio.
<ul style="list-style-type: none"> • <i>registraEvento(TipoServicio,TipoEstadoEjecucion): void</i>
Verifica si ya se ejecuto el servicio de conciliación del control numérico y devuelve true si fue correcto y false de no haberse ejecutado o suceder algún error.
<ul style="list-style-type: none"> • <i>verificaEjecucionConciliacionCtrnum():Boolean</i>
Verifica si ya se ejecuto el servicio de carga de datos de Dalí y devuelve true si fue correcto y false de no haberse ejecutado o suceder algún error.
<ul style="list-style-type: none"> • <i>verificaEjecucionDali():Boolean</i>
Verifica si ya se ejecuto el servicio de carga de datos del control numérico y devuelve true si fue correcto y false de no haberse ejecutado o suceder algún error.
<ul style="list-style-type: none"> • <i>verificaEjecucionControl():Boolean</i>

Tabla 7. Principales Métodos del servicio de Eventos.

3.5.1.2.3. NotificacionService.

Contiene todos los servicios relacionados a las notificaciones por correo electrónico, sus principales métodos se muestra en la tabla 8:

Cierra una notificación la cual fue previamente confirmada.
<ul style="list-style-type: none"> • <i>cierraNotificacion(Long):void</i>
Confirma una notificacion abierta.
<ul style="list-style-type: none"> • <i>confirmaNotificacion(Long,String):void</i>
Consulta todas las notificaciones del sistema
<ul style="list-style-type: none"> • <i>consultaNotificaciones():List</i>

consulta notificaciones por un rango de fechas y estatus determinado
<ul style="list-style-type: none"> <i>consultaNotificacionesByFechaEstatus(Date, Date, int):List</i>
Envía una notificación
<ul style="list-style-type: none"> <i>enviaNotificacion(NotificacionVO):void</i>
Envía un conjunto de notificaciones
<ul style="list-style-type: none"> <i>enviaNotificaciones(List):void</i>
Guarda una nueva notificación
<ul style="list-style-type: none"> <i>guardaNotificacion(NotificacionVO):void</i>
Realiza una búsqueda de todas las notificaciones abiertas del sistema y las reenvía.
<ul style="list-style-type: none"> <i>reenviaNotificaciones():void</i>

Tabla 8. Principales Métodos del servicio de Notificación.

3.5.1.2.4. ConciliaPartidaService

Contiene todos los servicios relacionados a la conciliación de las posiciones, expone servicios para realizar la conciliación, y obtener posiciones y movimientos previamente cargados en el sistema, sus principales métodos son:

Realiza el proceso de conciliación determinado por el nombre de la bóveda a conciliar.
<ul style="list-style-type: none"> <i>conciliarPartida(String)</i>
Busca un movimiento por el id de la posición y la bóveda a la que pertenece.
<ul style="list-style-type: none"> <i>findMovimientosByIdPosicionAndBoveda(Long, String):List</i>
Busca un movimiento determinado por su tipo valor, emisora y serie.
<ul style="list-style-type: none"> <i>findMovimientosByTVES(String, String, String, String):List</i>
Busca una posición por su Id
<ul style="list-style-type: none"> <i>findPosicionById(Long):PosicionVO</i>
Busca todas las posiciones que fueron conciliadas el día de hoy pasando como parámetro la bóveda a la que pertenecen.

- `findPosicionesConciliadas(String)`

Tabla 9. Principales Métodos del servicio de conciliación.

3.5.1.2.5. InformacionService

Contiene toda la lógica de extracción de datos desde Fuentes externas como el control numérico y Dalí, sus principales métodos se muestran en la tabla 10:

Realiza la extracción de posiciones y movimientos desde el control numérico.
<ul style="list-style-type: none"> • <code>registraInformacionControl():void</code>
Realiza la extracción de posiciones y movimientos desde Dalí.
<ul style="list-style-type: none"> • <code>registraInformacionDali():void</code>

Tabla 10. Principales Métodos del servicio de Información.

3.6. Vista de Información.

Se centra en las clases de información tratadas por el sistema, su semántica, y las restricciones impuestas sobre la utilización e interpretación de dicha información.

3.6.1. Modelo Entidad Relación.

El modelo de datos más extendido es el denominado ENTIDAD/RELACIÓN (E/R) En el modelo E/R se parte de una situación real a partir de la cual se definen entidades y relaciones entre dichas entidades.

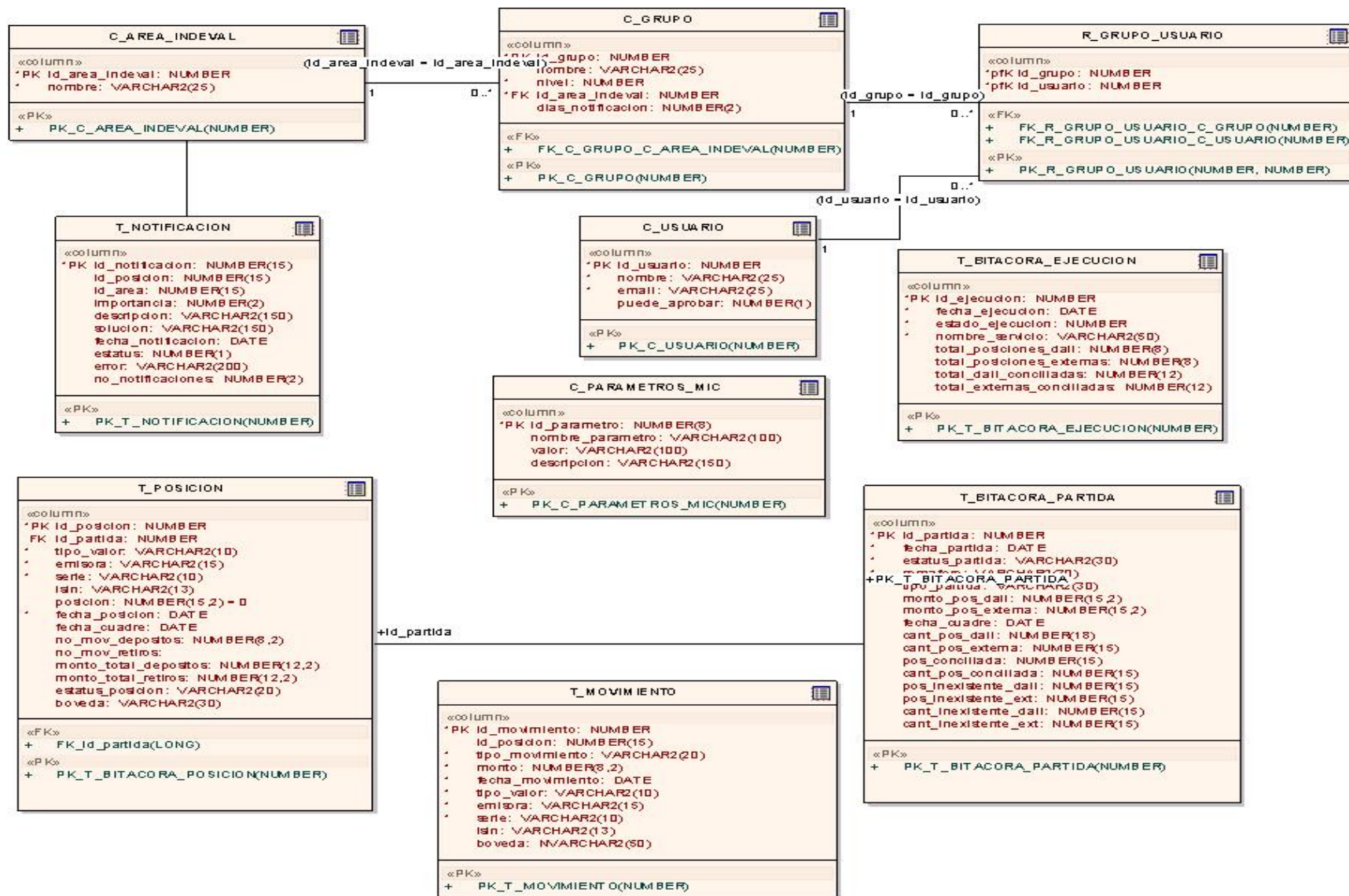


Figura 25. Modelo Entidad Relación del Sistema MIC

3.6.2. Catálogo de Elementos.

C_AREA_INDEVAL

El catálogo de las áreas de Indeval a las que se van a realizar las notificaciones se muestra en la tabla 11:.

CAMPO	DESCRIPCIÓN
ID_AREA_INDEVAL	Identificador único de cada una de las áreas de Indeval
NOMBRE	Nombre del área.

Tabla 11. Descripción de campos para C_AREA_INDEVAL.

C_GRUPO

El Catálogo de los grupos de usuarios en los que se divide un área se muestra en la tabla 12.

CAMPO	DESCRIPCIÓN
ID_GRUPO	Identificador único de cada uno de los grupos de Indeval
NOMBRE	Nombre del grupo.
NIVEL	Corresponde a un número el cual representa un nivel que tiene un grupo dentro del área correspondiente siendo 1 el nivel más bajo.
ID_AREA_INDEVAL	Identificador del área a la que pertenece el grupo.

Tabla 12. Descripción de campos para C_GRUPO.

C_USUARIO

El catálogo de los usuarios del sistema se muestra en la tabla 13.

CAMPO	DESCRIPCIÓN
ID_USUARIO	Identificador único de cada usuario.
NOMBRE	Nombre del usuario que recibirá las notificaciones.
EMAIL	Correo electrónico interno al cual se le enviará la notificación.
PUEDE_APROBAR	Campo que indica si un usuario puede confirmar que una notificación ha sido resuelta.

Tabla 13. Descripción de campos para C_USUARIO.

R_GRUPO_USUARIO

Tabla donde se guarda la relación de muchos a muchos para grupos y usuarios

CAMPO	DESCRIPCION
ID_GRUPO	Identificador único del grupo al que está relacionado el usuario
ID_USUARIO	Identificador único del usuario al que está relacionado el grupo

Tabla 14. Descripción de campos para R_GRUPO_USUARIO.

T_NOTIFICACION

Tabla donde se guardan todas las notificaciones del sistema:

CAMPO	DESCRIPCIÓN
ID_NOTIFICACION	Identificador único de cada una de las notificaciones
ID_POSICION	Identificador de la posición a la cual está asociada una notificación
ID_AREA	Identificador del área a la que está relacionada el área
IMPORTANCIA	Indica la importancia de la notificación, los valores que puede tener este campo son: 0 NORMAL, 1 BAJA, 2 MEDIA, 3 ALTA, 4 URGENTE
DESCRIPCION	Corresponde a la descripción que realiza el administrador del sistema sobre la posible causa del descuadre en las posiciones.
SOLUCION	En este campo se registra la solución al problema que dio origen a la notificación.
FECHA_NOTIFICACION	Fecha en la cual se creó la notificación.
ESTATUS	Estatus en el cual puede estar una notificación: 0 ABIERTO, 1 CONFIRMADO, 2 CERRADO
ERROR	En este campo se registra el tipo valor emisora, serie e isin de la posición que tiene una incidencia
NO_NOTIFICACIONES	Es el número de veces que se ha enviado una notificación.

Tabla 15. Descripción de campos para T_NOTIFICACION.

C_PARAMETROS_MIC

Tabla donde se guardan valores que sirven como parámetros del sistema, la estructura de los parámetros está dada por el nombre del parámetro y valor.

CAMPO	DESCRIPCIÓN
ID_PARAMETRO	Identificador único del parámetro
NOMBRE_PARAMETRO	Nombre del parámetro
VALOR	Valor del parámetro
DESCRIPCION	Descripción del parámetro

Tabla 16. Descripción de los campos de C_PARAMETROS_MIC.

T_POSICION

En la tabla 17 es donde se guardan todas las posiciones extraídas cada día de las fuentes de información.

CAMPO	DESCRIPCIÓN
ID_POSICION	Identificador único de la posición
ID_PARTIDA	Identificador única de la partida a la que pertenece la posición
TIPO_VALOR	Tipo valor de la posición
EMISORA	Emisora
SERIE	Serie
ISIN	Clave Isin
POSICION	Posición de la emisión
FECHA_POSICION	Fecha en la que se cargo la posición
FECHA_CUADRE	Fecha en la cual la posición se resolvió

NO_MOV_DEPOSITOS	Número total de movimientos de depósitos que tiene la posición
NO_MOV_RETIROS	Número total de movimientos de retiro que tiene la posición
MONTO_TOTAL_DEPOSITOS	Monto total correspondiente a todos los depósitos
MONTO_TOTAL_RETIROS	Monto total correspondiente a todos los retiros.
ESTATUS_POSICION	En este campo se guarda el resultado de la conciliación de la posición, el estatus puede ser DIFERENCIA_POSICION cuando al comparar dos posiciones existe una diferencia o INEXISTENTE cuando la posición no se encontró en la bóveda.
BOVEDA	Bóveda a la que pertenece la posición.

Tabla 17. Descripción de los campos de T_POSICION.

T_MOVIMIENTO

En la tabla 18 es donde se guardan todos los movimientos de las posiciones extraídos diariamente de las fuentes de información.

CAMPO	DESCRIPCIÓN
ID_MOVIMIENTO	Identificador único del movimiento.
ID_POSICION	Identificador de la posición a la cual pertenece el movimiento.
TIPO_MOVIMIENTO	Tipo de movimiento DEPOSITO o RETIRO.
MONTO	Monto del movimiento.
FECHA_MOVIMIENTO	Fecha en la que se realizó el movimiento.
EMISORA	Emisora.
TIPO_VALOR	Tipo valor.
SERIE	Serie.
ISIN	ISIN.

BOVEDA	Bóveda.
--------	---------

Tabla 18. Descripción de los campos de T_MOVIMIENTO.**T_BITACORA_PARTIDA**

Tabla donde se guarda un resumen de cada partida conciliada al final del día.

CAMPO	DESCRIPCIÓN
ID_MOVIMIENTO	Identificador único del movimiento.
ID_POSICION	Identificador de la posición a la cual pertenece el movimiento
TIPO_MOVIMIENTO	Tipo de movimiento DEPOSITO o RETIRO
MONTO	Monto del movimiento
FECHA_MOVIMIENTO	Fecha en la que se realizó el movimiento
EMISORA	Emisora
TIPO_VALOR	Tipo valor
SERIE	Serie
ISIN	ISIN
BOVEDA	Bóveda

Tabla 19. Descripción de los campos de T_BITACORA_PARTIDA.

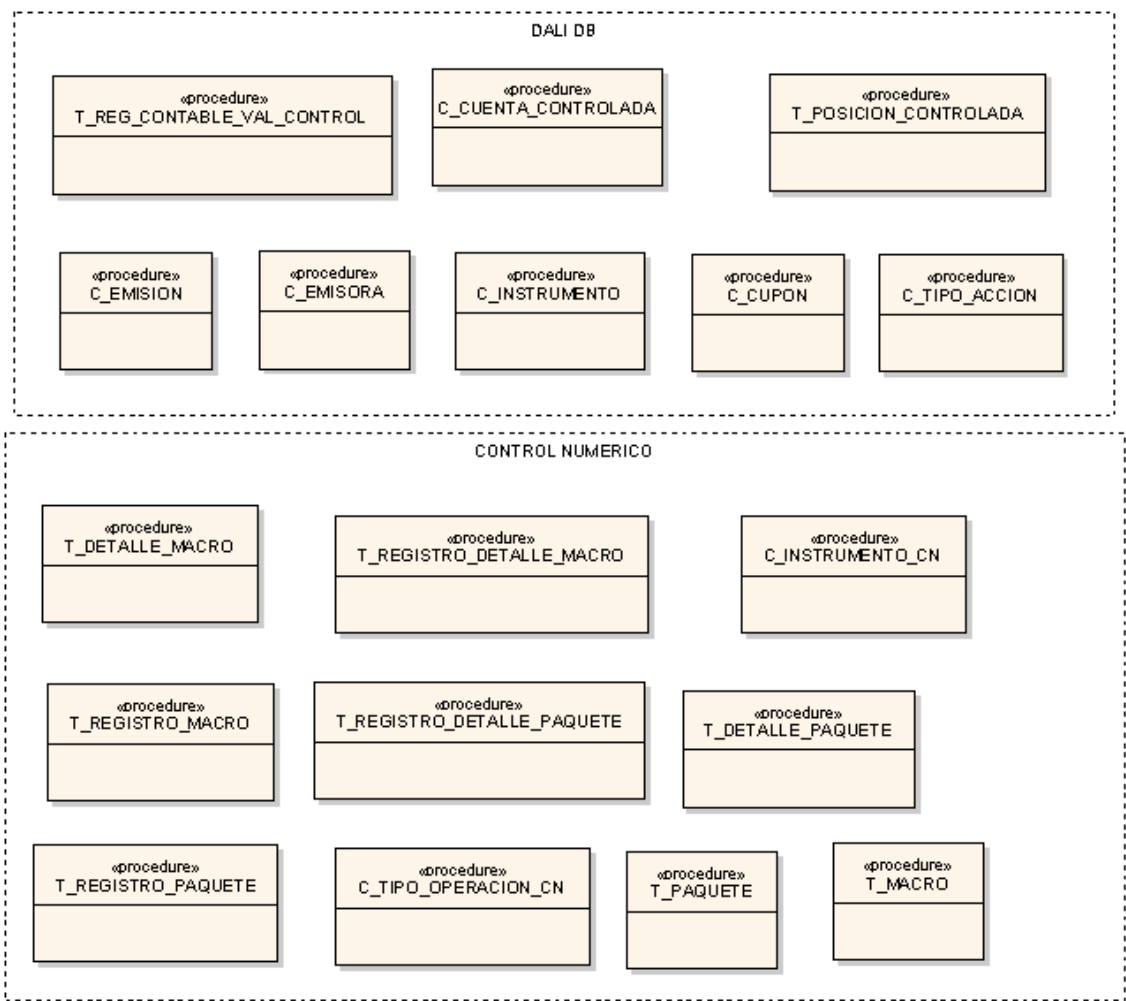


Figura 26. Estructura de las fuentes de datos externas.

Las fuentes de datos externas, actualmente están compuestas por dos fuentes donde se extraen los datos, la base de datos de Dalí y la base de datos del control numérico, en la figura 27 se muestran las entidades de las cuales se obtiene la información de forma directa.

Existe un tema con respecto a la estructura donde se guardan los estados y las prioridades correspondientes a las notificaciones que se envían dentro del sistema. Las figuras 28 y 29 muestran estos estados así como el ciclo de vida de las notificaciones:

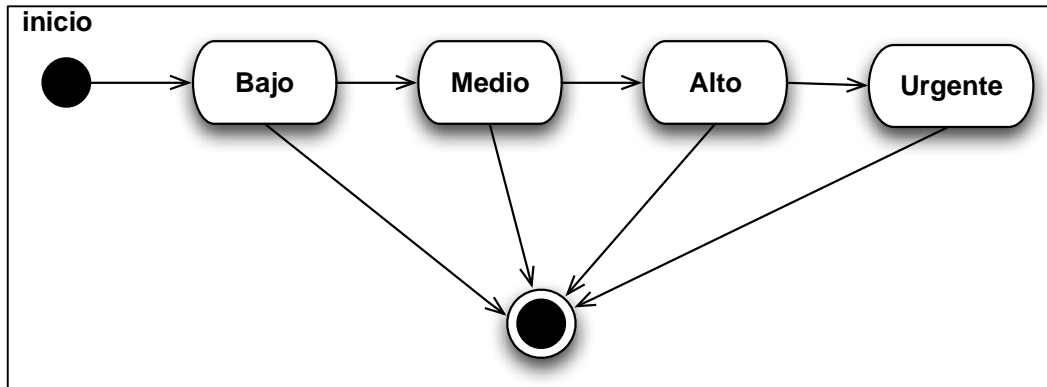


Figura 27. Transiciones de las prioridades de una notificación.

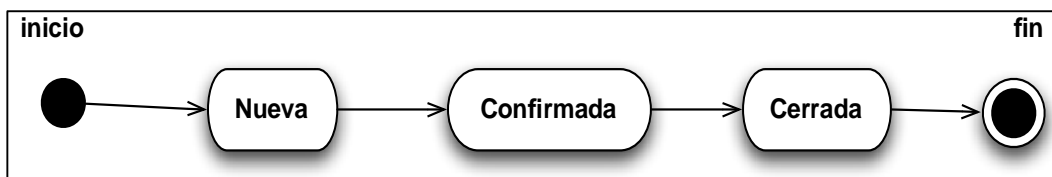


Figura 28. Ciclo de vida de una notificación.

3.7. Vista de Desarrollo.

La documentación de la vista normalmente incluye una representación gráfica de la estructura que se está modelando. Aunque existe libertad con respecto a la notación usada, lo más común y conveniente es usar UML.

3.7.1. Presentación Inicial.

El desarrollo del proyecto requiere del conocimiento de algunos marcos de trabajo de terceros como *Spring* para la interconexión entre componentes y *Hibernate* para la persistencia de datos, la aplicación se encuentra dividida de forma lógica en cuatro capas como se muestra a continuación en la figura 30:

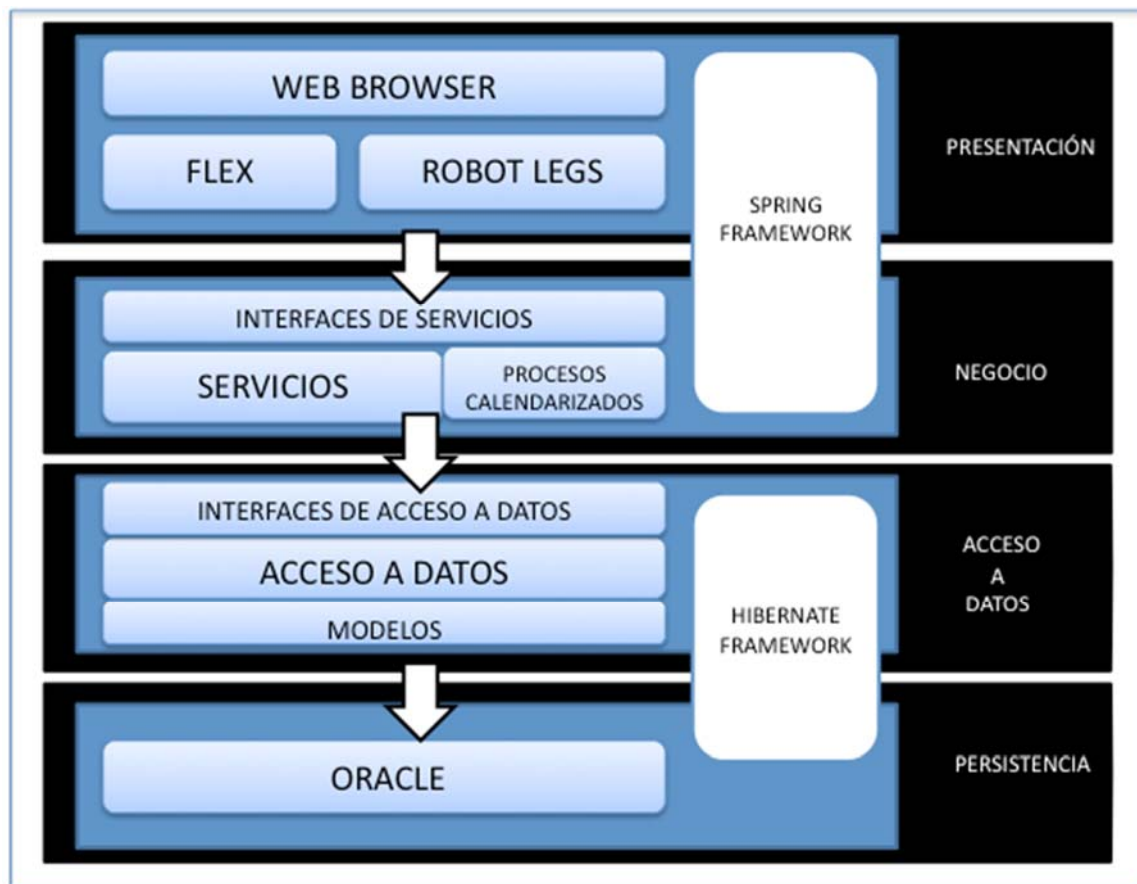


Figura 29. Estructura de capas del MIC.

3.7.1.1. Capa de Presentación.

El principal componente de la capa de presentación es el marco de trabajo de *Robot Legs* el cual está basado en un estilo de arquitectura de software denominado Modelo Vista

Controlador (MVC). Dicho modelo separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Para *Robot Legs* la clase *Context* es el corazón del marco de trabajo, sirve como un *bus* centralizado de eventos y permite que las implementaciones de las diferentes partes de la arquitectura se comuniquen entre sí. Es el mecanismo de arranque que inicializa la inyección de dependencias y otras utilidades del marco de trabajo, también define ámbito, los actores del marco de trabajo viven dentro de un contexto y se comunican con otros que viven dentro del mismo ámbito de ese contexto, en una aplicación pueden existir varios contextos, estos pueden comunicarse entre sí, lo cual es muy útil para el desarrollo de aplicaciones modulares que requieren comunicación entre módulos.

La capa controladora se representa por la clase *Command*. Los comandos son objetos de corta vida, sin estado, que representan acciones individuales a ejecutar por la aplicación, comúnmente como respuesta a interacciones con el usuario pero su uso no está limitado a esto. Estos objetos pueden enviar eventos que pueden ser recibidos por mediadores para ejecutar acciones sobre las vistas, o que pueden desencadenar la ejecución de otros comandos.

La capa de la vista es representada por la clase *Mediator*. Las clases que heredan de *Mediator* se utilizan para controlar la interacción de los actores del con los componentes visuales. Los mediadores escuchan los eventos despachados por los actores del marco de trabajo y en consecuencia modifican el estado del componente visual que representan, registran escuchadores de los eventos generados por el componente visual que representan y despachan además eventos como respuesta éstos. Esto facilita poner la lógica del componente visual en el

mediador y desacoplarlo al máximo de la aplicación, permitiendo su reutilización y modificación con el menor costo.

Para implementar la capa del modelo y de los servicios el marco de trabajo ofrece la clase *Actor*. Cualquier clase que herede de *Actor* tiene la funcionalidad de comunicarse con el resto de los actores del marco de trabajo.

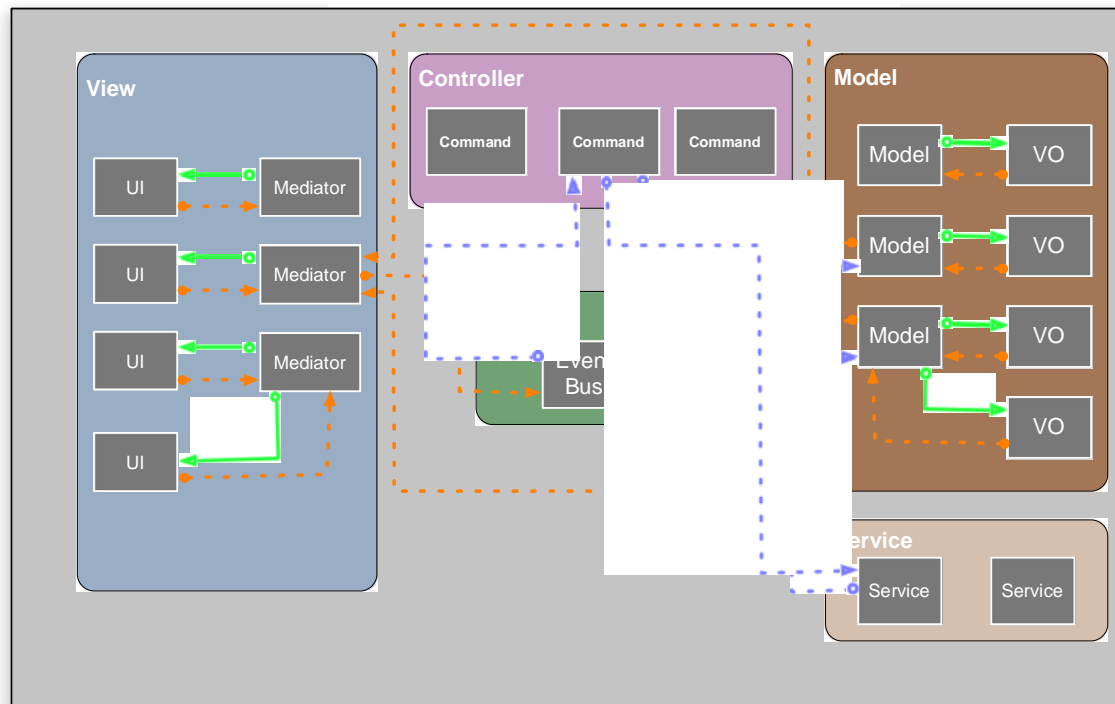


Figura 30. Marco de trabajo para la capa de presentación.

3.7.1.1.1. Principales Reglas.

- Todos los objetos VO (*value objects*) son mapeados a un objeto VO de Java perteneciente a la capa de acceso a datos.
- Todos los componentes mxml de la capa deben tener asociados una hoja de estilos correspondientes a los estilos predefinidos para el sistema.
- Por cada servicio a utilizar expuesto por la aplicación Java debe de existir un servicio del lado de la capa de presentación.
- Por ser una aplicación mediana, solo existirá un *context* para toda la aplicación donde residirá la lógica de los servicios a llamar, en caso de que se necesite agregar un contexto adicional posteriormente, se tendrá que documentar el motivo por el que se añadió y modificar esta sección del documento.
- Dentro de la aplicación existe un paquete denominado *util* donde se almacenaran todas las utilerías del sistema, validaciones comunes a diferentes componentes y formateadores, adicionalmente a esto existirá otra carpeta denominada *components* donde residirán los componentes creados para la aplicación.

3.7.1.1.2. Nomenclatura de los componentes.

Todo componente dentro de la capa de presentación debe seguir las siguientes reglas de nomenclatura de los componentes:

- El nombre de los componentes comenzará con mayúsculas utilizando la notación camel para nombrar las clases.

- Los componentes de tipo *Command* deben de ser nombrados de acuerdo a la acción que realiza el comando, por ejemplo si se requiere borrar un usuario se llamará al comando `BorrarUsuarioCommand` donde la primera parte del nombre del componente se refiere a la acción y posteriormente se agrega la palabra *Command*.
- Los servicios se nombraran de acuerdo al servicio que están prestando y si es la implementación del servicio o la interfaz la que se está nombrando para interfaces se agregará al final la palabra *Service*, por lo que si se requiere agregar la interfaz de un servicio para consultas de usuarios el servicio se llamara `ConsultasUsuariosService` y la implementación de esta se llamará `ConsultasUsuariosServiceImpl`.
- Los componentes de tipo mediadores se les agregará al final del nombre la palabra *ViewMediator*
- Los componentes de tipo evento se les agregará al final del nombre la palabra *Event*.

3.7.2. Capa de Negocio

Dentro de esta capa se encuentra el centro de la aplicación, es donde se exponen los servicios principales del sistema, esta capa utiliza el marco de trabajo de *Spring* para poder ligar todas las dependencias entre los componentes mediante archivos xml.

Los principales componentes de esta capa son los servicios expuestos:

- `CatalogoService`
- `ConciliaPartidaService`
- `InformacionService`

- `NotificacionService`
- `BitacoraEventosService`

3.7.2.1. Reglas.

- a. Los servicios solo pueden utilizar recursos de la capa de acceso a datos, estos servicios no podrán utilizar recursos de capas inferiores a la de acceso a datos y serán proveedores de las capas superiores.
- b. Todo servicio a exponer se debe de realizar mediante una interfaz y su utilización debe de ser mediante un archivo de contexto xml.
- c. Las reglas de negocio de la aplicación deben de estar expresadas en estos componentes, cualquier regla o validación que no tenga que ver con las reglas de negocio del sistema queda fuera de esta capa para su inclusión en otras capas.

3.7.3. Capa de Acceso a datos.

Esta capa contiene clases que implementan la lógica de acceso a datos, en esta capa se encuentran objetos intermedios entre la capa de persistencia dentro del cual se manejan las reglas correspondientes al acceso a datos

Las reglas a seguir dentro de esta capa son:

Las clases deben de terminar con la palabra *Dao* (*Data Access Object*) para las interfaces y *DaoImpl* para las implementaciones.

Toda la lógica para la construcción de consultas se debe de realizar en esta capa.

Como regla dentro del sistema, para las consultas que se utilizan para la extracción de datos de entidades externas al sistema, se utiliza el acceso a base de datos directamente sin pasar por los modelos, esto sólo para los sistemas internos de Indeval. Para el acceso a los componentes del sistema se realiza llamando a una capa extra denominada modelos, en donde se mapean directamente las tablas de la base de datos a objetos.

3.7.4. Capa de Persistencia.

Dentro de esta capa se encuentran las clases que mapean directamente objetos a tablas de la base de datos y se crea un modelo por cada tabla de la base de datos a utilizar. Para esta capa es necesario que los desarrolladores tengan conocimiento de bases de datos relacionales así como las herramientas que provee el marco de trabajo de *Hibernate* para la correcta construcción de consultas y relaciones de los objetos.

3.7.4.1. Notación.

Indentación

Se deberían usar cuatro espacios como unidad de indentación. La construcción exacta de la indentación no está especificada (espacios o tabs). Los tabuladores deben ser exactamente cada 8 espacios (no cada 4).

Longitud de Línea.- Evitar líneas mayores de 80 caracteres, ya que no son bien manejadas por muchos terminales y herramientas.

Ruptura de Líneas.- Cuando una expresión no entre en una sola línea, se debe romper de acuerdo a estos principios generales:

- Romper después de una coma.
- Romper antes de un operador
- Preferir las rupturas de alto nivel a las de bajo nivel.
- Alinear la nueva línea con el principio de la expresión al mismo nivel de la línea anterior.

Si las reglas anteriores conducen a la confusión del código o código que se estrella contra el margen derecho, sólo indentamos 8 espacios.

Inicialización.- Debemos intentar inicializar las variables locales donde son declaradas. La única razón para no inicializar una variable donde es declarada es si el valor inicial depende de algún cálculo que tiene que ocurrir antes.

Sentencias if, if-else, if else-if else

Las sentencias del tipo *if-else* deberían tener la siguiente forma:

```
if ( condition) {      statements; } if ( condition) {      statements;  }  else  {  
    statements; } if ( condition) {      statements;  }  else  if ( condition) {  
    statements; } else {  statements; }
```

Líneas en Blanco.- Las líneas en blanco mejoran la lectura separando secciones de código que están relacionadas lógicamente.

Siempre se deberían usar dos líneas en blanco en las siguientes circunstancias:

- Entre secciones de un fichero fuente
- Entre definiciones de clases e interfaces

Siempre se debería usar una línea en blanco en las siguientes circunstancias:

- Entre métodos.
- Entre las variables locales de un método y su primera sentencia.
- Antes de un bloque de comentarios o un comentario simple.
- Entre secciones lógicas dentro de un método para mejorar su lectura.

3.7.5. Convenciones de Nombrado.

Las convenciones de nombrado hacen los programas más entendibles haciéndolos más fáciles de leer. También pueden proporcionar información sobre la función del identificador, por ejemplo, si es una constante, un paquete o una clase, lo que puede ayudarnos a entender el código, las convenciones mostradas en la tabla 20 son las utilizadas dentro del proyecto.

Tipo de Identificador	Reglas de Nombrado	Ejemplos
Paquetes	Los prefijos de un nombre de paquete único siempre se escriben en letras ASCII minúsculas y deberían ser uno de los nombres de dominios de alto nivel, actualmente com, edu, gov, mil, net, org, o uno de los códigos de dos letras que identifican los países como especifica el estándar ISO 3166, 1981. Los siguientes componentes del	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

	<p>nombre de paquete varían de acuerdo a las convenciones de nombrado internas de una organización. Dichas convenciones podrían especificar que ciertos nombres de componentes directorio serían división, departamento, proyecto, máquina, o nombres de <i>login</i>.</p>	
Clases	<p>Los nombres de clases deben ser mezclas de mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúsculas. Debemos intentar mantener los nombres de clases simples y descriptivos. Debemos usar palabras completas y evitar acrónimos y abreviaturas (a menos que la abreviatura se use muy ampliamente como URL o HTML).</p>	<pre>class Raster; class ImageSprite;</pre>
Interfaces	<p>Los nombres de interfaces se tratan igual que los nombres de clases</p>	<pre>interface RasterDelegate; interface Storing;</pre>
Métodos	<p>Los métodos deberían ser verbos, en mayúsculas y minúsculas con la primera letra del nombre en minúsculas, y con la primera letra de cada palabra interna en mayúsculas.</p>	<pre>run(); runFast(); getBackground();</pre>
Variables	<p>Las variables, tanto de ejemplar, de clase, como las constantes de clase se escriben en mayúsculas y minúsculas y con la primera letra del nombre en minúsculas, y con la primera letra de cada palabra interna en mayúsculas. Los nombres de variables no deben empezar con los caracteres subrayado "_" o dólar "\$", incluso aunque estén permitidos.</p> <p>Los nombres de variables deberían ser cortos y llenos de significado. La elección de una variable debería ser mnemónica-es decir, diseñada para indicar al observador casual su utilización. Se deben evitar los</p>	<pre>int i; char c; float myWidth;</pre>

	nombres de variable de un sólo caracter, excepto para variables temporales. Algunos nombres comunes de este tipo de variables son: i, j, k, m, y n para enteros.	
Constantes	Los nombres de variables constantes de clases y las constantes ANSI deberían escribirse todo en mayúsculas con las palabras separadas por subrayados (“_”). (Se deberían evitar las constantes ANSI para facilitar la depuración.)	<pre>static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999; static final int GET_THE_CPU = 1;</pre>

Tabla 20. Convenciones de nombrado de código

3.8. Vista de Operación.

Describe cómo el sistema será operado, administrado y soportado en el entorno de producción.

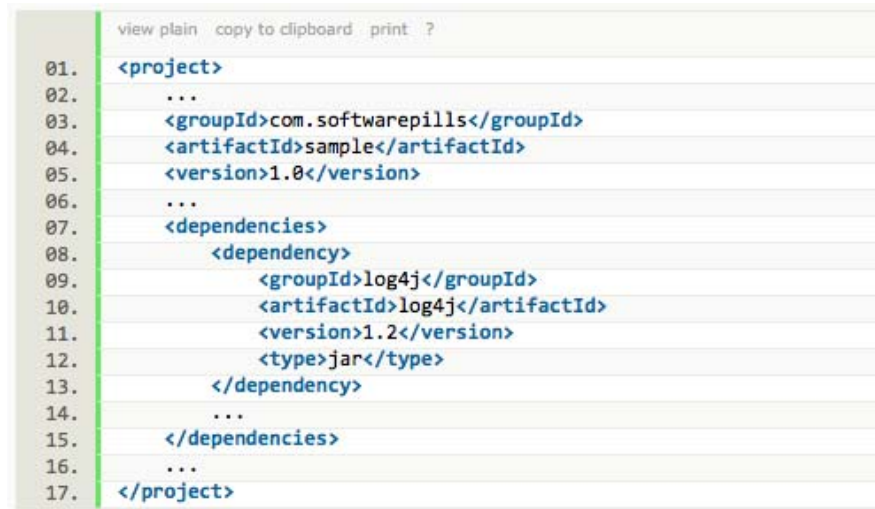
3.8.1. Instalación del sistema.

Los productos que se generan para la instalación de la aplicación son los siguientes:

- *War* correspondiente a la aplicación web.
- *War* correspondiente a los procesos calendarizados del sistema.

Maven utiliza un *Project Object Model* (POM) el cual es un archivo que se usa para describir el proyecto de *software* a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Para especificar las dependencias en Maven se utiliza la sección *dependency* del POM mostrado en la figura 32:



```
view plain copy to clipboard print ?
01. <project>
02.   ...
03.   <groupId>com.softwarepills</groupId>
04.   <artifactId>sample</artifactId>
05.   <version>1.0</version>
06.   ...
07.   <dependencies>
08.     <dependency>
09.       <groupId>log4j</groupId>
10.       <artifactId>log4j</artifactId>
11.       <version>1.2</version>
12.       <type>jar</type>
13.     </dependency>
14.     ...
15.   </dependencies>
16.   ...
17. </project>
```

Figura 31. Estructura de un archivo POM.

La generación de los productos anteriores se realiza mediante la herramienta de *Maven* utilizando el siguiente comando:

```
Mvn -Dmaven.test.skip=true clean install
```

Esto creará un directorio *target* justo debajo de la carpeta del proyecto y ahí un subdirectorio *classes* donde meterá todos los *.class* de nuestro compilado como se muestra en la figura 33.

```

EjemploMaven
+---src
|   +---main
|   |   +---java
|   |   |   +---chuidiang
|   |   |   +---ejemplos
|   +---test
|       +---java
|       +---chuidiang
|       +---ejemplos
+---target
    +---classes
        +---chuidiang
        +---ejemplos    //Aquí van los ficheros .class

```

Figura 32.. Ejemplo de estructura de directorios creados con maven.

3.8.2. Monitoreo de la aplicación.

El monitoreo de la aplicación se puede realizar de diferentes formas:

- **Monitoreo desde el servidor de aplicaciones.-** Dentro del servidor de aplicaciones se puede monitorear el estatus de las aplicaciones para saber si estas se encuentran en línea, este sería el primer lugar a revisar cuando la aplicación no esté en línea.
- **Monitoreo de los logs escritos por la aplicación.-** Todos los mensajes que escribe la aplicación a los logs tienen un nivel de prioridad, de acuerdo al nivel de prioridad establecido para el mensaje y al nivel configurado en el archivo de configuración estos mensajes se podrán mostrar o no.

Por defecto Log4J tiene 6 niveles de prioridad para los mensajes (trace, debug, info, warn, error, fatal). Además existen otros dos niveles extras (all y off):

Niveles de prioridad (De mayor -poco detalle- a menor -mucho detalle-):

- a) FATAL: se utiliza para mensajes críticos del sistema, generalmente después de guardar el mensaje el programa abortará.
- b) ERROR: se utiliza en mensajes de error de la aplicación que se desea guardar, estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.
- c) WARN: se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan al correcto funcionamiento del programa.
- d) INFO: se utiliza para mensajes similares al modo "verbose" en otras aplicaciones.
- e) DEBUG: se utiliza para escribir mensajes de depuración. Este nivel no debe estar activado cuando la aplicación se encuentre en producción.
- f) TRACE: se utiliza para mostrar mensajes con un mayor nivel de detalle que debug.
- g) ALL: este es el nivel de máximo detalle, habilita todos los logs (en general equivale a TRACE).
- h) OFF: este es el nivel de mínimo detalle, deshabilita todos los logs.

Loj4j trae consigo un monitor de mensajes denominado chainsaw, hecho con java el cual permite ver el contenido de aquellos archivos de log en formato XML.

De no tener la herramienta chainsaw instalada y configurada correctamente los archivos de log se pueden abrir con cualquier editor de texto para ser analizados.

Revisando las tablas de la base de datos.- Se pueden realizar consultas para ver el estatus de la operación del sistema, las principales operaciones que se pueden llevar a cabo para ver el correcto funcionamiento de la aplicación son:

Revisión de la tabla de t_bitacora_ejecucion dentro de la cual debe de estar un registro para cada una de las extracciones, adicionalmente un registro mas cuando se ejecuta la conciliación, por lo que si no existen estos registros podemos dar por hecho que no se han ejecutado estos servicios o que pudiera haber un error al ejecutarse por lo que la aplicación puede presentar fallos.

Revisión de la tabla de t_posicion dentro de la cual debe de haber registros de las posiciones pertenecientes a cada una de las bóvedas, las posiciones se cargan de forma diaria, al momento de ejecutarse los servicios de extracción de datos si no llegase a encontrar posiciones, la conciliación no podría ejecutarse para ese determinado día.

Revisión de la tabla de t_notificacion dentro de esta tabla se encuentran todas las notificaciones generadas por el sistema, estas notificaciones se pueden manipular cambiando su estatus, una notificación se puede volver a activar siempre y cuando su estatus no sea de cerrada y la partida a la que pertenece este abierta de lo contrario la notificación ya no se volverá a enviar.

CAPITULO IV

Resultados

Después de haber realizado el caso de práctica y su documentación estamos en condiciones de describir los resultados derivados y formular las conclusiones definitivas.

Analizando el primero de los objetivos que se planteo al inicio podemos identificar una serie de actividades que pueden suceder dentro de la operación de Indeval, con respecto al caso práctico y que se pueden solucionar de una forma más ordenada teniendo la documentación de la arquitectura:

- a) Los usuarios que dan mantenimiento al sistema se han cambiado de área y nuevos usuarios se tendrán que incorporar para dar el servicio.
- b) Se necesita realizar un cambio en la lógica de extracción de las posiciones de Indeval ya que se modificaron los campos de las tablas donde se extraen.
- c) Un nuevo equipo de desarrollo realizará un nuevo módulo para incluir unos procesos de aprobación de las notificaciones.

Analizando el inciso a, y basándonos en lo que tenemos hasta este momento, podemos referir a los usuarios a:

- Leer la descripción general del sistema.
- Leer la vista de distribución.

Con lo anterior se cubre una parte importante ya que de lo contrario los usuarios deberán tener un mayor tiempo de inducción al negocio y capacitación del sistema. La documentación no debería suprimir estos temas, sin embargo sí reducirá considerablemente los tiempos para exponerlos y asegurará que la información que se está proporcionando sea congruente con el sistema.

Para resolver el punto b, al estar relacionado este nuevo requerimiento con datos dentro del sistema, podemos realizar las siguientes tareas:

- Verificar en la vista de Información si para la estructura actual de los datos deberán realizarse cambios.
- Dentro de la vista de información se puede ver que este requerimiento está asociado al componente de extracción de la información.
- En el diagrama de estructura de capas de la vista de desarrollo podemos apreciar que el cambio que se requiere se encuentra en el acceso a datos y que posiblemente pueda afectar los modelos de persistencia.
- Una vez que se analizó la información anterior, los cambios a realizar se encuentran bien localizados y es más fácil estimar el esfuerzo necesario tanto en tiempo como en recursos para resolverlo.

Si un nuevo equipo de desarrollo necesita realizar un nuevo requerimiento para el sistema, el documento de arquitectura sería como un mapa a seguir para analizar cómo fue que se construyó el sistema actual, y ver en qué parte y de qué forma se integrará el

nuevo modulo, si la nueva funcionalidad vive dentro de los componentes existentes o si se tienen que agregar nuevos al diseño.

Sin este documento el nuevo equipo tendría que realizar una serie de técnicas como la ingeniería inversa, entrevistas con el equipo que lo desarrolló, en caso de estar disponible, y lectura del código fuente, para poder obtener las reglas de negocio así como analizar su estructura para determinar cuáles fueron los estándares a utilizar, realizar entrevistas con los usuarios para verificar y validar las reglas de negocio obtenidas. Esto haría que se requiriera más tiempo para realizar los cambios y obtener la nueva funcionalidad con la calidad esperada.

Dentro de Indeval se determinó que para que la documentación de la arquitectura cumpla con el estándar, se deben seguir las siguientes prácticas:

- Identificación del personal involucrado y sus intereses. Se describen los diversos tipos de personas involucradas en el proyecto. Tales como, administradores, diseñadores, desarrolladores, usuarios, patrocinadores. A su vez, se incluye la diversidad de intereses que la arquitectura debe satisfacer.
- Puntos de vista. Cada uno de estos debe contener un nombre, personal involucrado, intereses que satisface, lenguaje o técnicas de modelado utilizados durante la construcción de la vista, y su justificación.
- Vistas. Cada vista debe tener un identificador, una breve introducción y la representación del sistema con respecto a un punto de vista en particular.

- Consistencia entre vistas. Registra las inconsistencias entre las vistas, así como algún tipo de procedimiento que indique la consistencia entre ellas.
- Justificación. Se debe incluir la justificación del tipo de arquitectura seleccionada, y de los puntos de vista utilizados.

Es importante mencionar que la idea de documentar las decisiones que se toman durante las actividades de diseño fue muy importante para Indeval. Esta documentación permite que posteriormente se evalúe el diseño además de que permite comprender la toma de decisiones del arquitecto al momento de realizar el mantenimiento del sistema.

CAPITULO V

Conclusiones.

Para poder concluir con este trabajo podemos tratar de contestar las preguntas planteadas al inicio

¿Por qué la documentación de la arquitectura es tan importante para el nuevo sistema de Indeval?

Para contestar la pregunta anterior podemos dar una serie de argumentos en los cuales basarnos para decir que es muy importante para Indeval y cualquier empresa el producir un documento que pueda mostrar la arquitectura de sus sistemas a cualquier persona interesada en conocerla.

Conocimiento unificado de todos los interesados en el sistema.

Al momento de tener la documentación correspondiente del sistema todos los usuarios pueden acceder a la misma información, pueden discutir sobre los diferentes aspectos del sistema sin provocar ambigüedad

Facilidad al realizar cambios.

Localizar los componentes a modificar dentro del sistema es más sencillo debido a que se puede realizar un análisis del tipo de cambio a realizar y los diferentes aspectos de la aplicación que puede llegar a impactar.

Menor tiempo de capacitación.

Toda persona interesada en conocer la arquitectura tiene un objetivo principal a conocer. El documento de arquitectura le permite a este tipo de usuarios el poder conocer de forma más precisa la parte que necesita sin necesidad de entender todas las partes que lo conforman.

Reutilización de técnicas, procedimientos y código en proyectos futuros.

Tener guías para la codificación y construcción del sistema es una gran ventaja para la empresa debido a que estas se pueden reutilizar en futuros proyectos, permite que varios proyectos estén basados en los mismos estándares y permite que los desarrolladores que modifican un sistema puedan verificar sistemas parecidos de forma más rápida.

Análisis de errores comunes y mejoras para los diferentes proyectos de la empresa.

Al usar los mismos procedimientos dentro de varios sistemas, las mejoras que se puedan realizar en un sistema se pueden propagar a los demás sistemas en componentes y procedimientos comunes. En caso de existir errores estos también se propagarían pero al arreglarlos para un sistema se corrigen para los demás componentes que comparten la funcionalidad.

Poder validar de forma fácil que se están cumpliendo correctamente las especificaciones iniciales del sistema.

En caso de querer realizar una auditoría para validar que el sistema está funcionando correctamente de acuerdo a las especificaciones iniciales de los *stakeholders* es fácil poder seguir estos requerimientos a través del documento.

¿Qué tipo de documentación es la que Indeval necesita?

La documentación que se realizó se basó en las necesidades de Indeval y del tipo de proyecto. Indeval necesitaba una documentación fácil de leer que reflejara los aspectos más importantes de acuerdo a los intereses de los *stakeholders* sin entrar a detalles muy específicos ya que es un sistema pequeño que sólo está desarrollado por una persona.

Referencias

- [Bass 03] L. Bass, P. Clements, and R. Kazman(2003). Software Architecture in Practice, 2nd ed., Reading, MA: Addison-Wesley.
- [ANSI/IEEE, 00] ANSI/IEEE Std 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems
- [Rozanski05]Rozanski, Nick and Eóin Woods, Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, Addison-Wesley Professional, 2005.
- [Kruchten 95] P. Kruchten(1995). "The 4+1 View Model of Architecture," IEEE Software, vol. 12, no. 6, pp. 45-50.
- [Clements 02] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord & Judith
- [Stafford 02]. "Documenting Software Architectures: Views and Beyond"., Addison Wesley Professional.
- [Clements01] Paul Clements,Rick Kazman,Mark Klein(2002). Evaluating Software Architectures Methods an case Studies, Addison-Wesley
- [Dijkstra68] Edsger Dijkstra. "GO-TO statement considered harmful". ACM Communications of the ACM, 11(3), pp. 147-148,Marzo de 1968.
- [SEI06] Software Engineering Institute, SEI. "Views and Beyond Architecture Documentation Template".,
- [Klein] Mark Klein; Rick Kazman & Robert Nord. "Attribute-Based Architectural Styles" Software Engineering Institute, Technical Report CMU/SEI-99-TR-22, Carnegie Mellon University, 1999
- [Brooks75] Frederick Brooks Jr. The mythical man-month. Reading, Addison-Wesley, 1975.
- [Parnas72] David Parnas. "On the Criteria for Decomposing Systems into Modules." Communications of the ACM 15(12), pp.1053-1058, Diciembre de 1972.
- [Perry92] Dewayne E. Perry y Alexander L. Wolf. "Foundations for the study of software architecture". ACM SIGSOFT Software Engineering Notes, 17(4), pp. 40–52, Octubre de 1992.
- [Shaw96] Mary Shaw y David Garlan. Software Architecture: Perspectives on an emerging discipline. Upper Saddle River, Prentice Hall, 1996.
- [Booch93] G. Booch: Object-Oriented Analysis and Design with Applications, 2nd. edition, Benjamin-Cummings Pub. Co., Redwood City, California, 1993, 589p.

[Garlan93] D. Garlan & M. Shaw, "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering, Vol. 1, World Scientific Publishing Co. (1993).

[Rubin92] K. Rubin & A. Goldberg, "Object Behavior Analysis," CACM, 35, 9 (Sept. 1992) 48-62