Universidad de las Ciencias Informáticas

Facultad 10

Proyecto de Informatización de La Casona 23yB.
Título: "Arquitectura General del Proyecto Centro Rector de Universidad para Todos."

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

Autor
Raúl Alejandro Fleitas Conill
Yoelkys Sabina Rodríguez

Tutor
Ing. Evelio Maikel Medina Manrique

Co-Tutor
Ing. Maykell Frómeta Flores

Ciudad de La Habana
Junio, 2007

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Firma del Autor Firma del Tutor _____ _____ _____

Raúl Fleitas Yoelkys Sabina Evelio Medina

AGRADECIMIENTOS

A nuestros padres por estar siempre junto a nosotros, por brindarnos todo su amor y apoyo incondicional durante todas nuestras vidas, en especial por todos sus esfuerzos para que este trabajo fuera una realidad.

A Maikel Frómeta Flores, que sin ninguna obligación y de manera desinteresada nos dedicó parte de su tiempo a orientarnos y guiarnos para poder empezar y culminar satisfactoriamente este trabajo.

A Misael y Edi por dedicarnos parte de su tiempo a aclararnos algunas dudas que fueron de gran utilidad.

A Daynel Mármol, por su ayuda en la tesis y todo su apoyo.

A todos nuestros amigos, que nos brindaron su ayuda, gracias por su apoyo y por sus contribuciones a este trabajo.

De Yoelkys:

A Raúl por trabajar incansablemente conmigo como un verdadero equipo para que este proyecto saliera adelante, a mi pareja Maidelys por brindarme su apoyo y su ayuda además de sus aportes a este trabajo.

De Raúl:

A Yoelkys por trabajar incansablemente conmigo como un verdadero equipo para que este proyecto saliera adelante, a mi pareja Layla por brindarme su apoyo y su ayuda desde mucho antes de comenzar este trabajo.

DEDICATORIA

De Yoelkys:
A mis padres: Jorge y Georgina.
A toda mi familia.

De Raúl:
A mis queridos padres Gustavo y Tomasa.
A mi único abuelo en vida Domingo.
A toda mi familia y personas cercanas.

RESUMEN

La Casona 23yB es una institución que fue creada al calor de la batalla de ideas. A esta institución asisten los profesores de Universidad para Todos y los Canales Educativos, con el objetivo de que dichos profesionales se reúnan para preparar los materiales necesarios para impartir diferentes cursos y organizar eventos que tienen que ver directamente con estos programas de la revolución. La Casona 23yB cuenta con una serie de recursos y servicios que posibilitan el apoyo necesario para desarrollar exitosamente todas las actividades que se prevean. La institución desea automatizar los servicios que ofrece para así brindar una atención de excelencia a los clientes que van a utilizar dichos servicios.

En este trabajo se propone una arquitectura de software que sirva de guía para desarrollar la automatización de La Casona 23yB, que va a realizarse mediante la elaboración de un sistema que funcionará sobre la base de Servicios Web. En el trabajo se presentan una serie de vistas arquitectónicas que dan descripción a la arquitectura así como distintos métodos para evaluar la misma, dicha arquitectura permitirá automatizar los principales servicios que brinda la institución ofreciendo una mejor calidad en los servicios.

ÍNDICE

INTRODUCCIÓN

La Casona 23yB se encuentra en la provincia de Ciudad de La Habana, es un local destinado a profesores de Universidad para Todos y Canales Educativos. La institución propone ser el lugar propicio para reunir a todos los profesores que han colaborado con estos grandes proyectos de La Batalla de Ideas; plantea convertirse en el centro donde dichos profesionales se reúnan y preparen los materiales necesarios para impartir diferentes cursos y organizar eventos; además contar con una serie de recursos y servicios que posibiliten el apoyo necesario para desarrollar exitosamente todas las actividades que se prevean.

Como consecuencia de que actualmente no existe ninguna operación que ayude al mejoramiento de la gestión y desarrollo del trabajo que se realiza en la institución, la **situación problémica** la constituye la falta de un sistema automatizado que regule el funcionamiento general de la institución, por lo que se plantea como objetivo lograr un sistema tecnológico que soporte y potencie el desarrollo de la institución, ya que una de las potencialidades previstas para esta es que aproveche las oportunidades que ofrecen las NTIC como factor decisivo que la define como una institución de nuevo tipo.

Por lo antes planteado La Casona 23yB decidió informatizar la institución para lograr la creación de un centro que aporte a la gestión del conocimiento de los profesionales que la utilizan y de proyectos tan importantes para el desarrollo intelectual del país como lo son Universidad para Todos y los cursos que se imparten por los Canales Educativos.

Por lo que el **problema** esta dado por esta interrogante: ¿Como construir un sistema automatizado para lograr un buen funcionamiento de La Casona 23yB, haciendo que esta sea una institución de nuevo tipo?

El **objeto de estudio** lo constituye la Arquitectura del software.

El **campo de acción** que abarca este trabajo es la creación de una Arquitectura Orientada a Servicios para sistema de La Casona 23yB.

Como **objetivo general** de este trabajo se propone crear una arquitectura de software que permita el desarrollo del sistema para La Casona 23yB.

Derivándose los siguientes **objetivos específicos:**

- Realizar un estudio de las arquitecturas más usadas en los sistemas de instituciones similares a La Casona 23yB.
- Realizar un estudio profundo de las características del funcionamiento del Proyecto Centro Rector de universidad para todos.

- Realizar un estudio de todas las tecnologías disponibles en el mercado sobre sistemas de identificación, sistema de control de información de usuarios, sistema de servicios de reservaciones y solicitudes, sistema de control de medios básicos y sistemas para la intranet para elegir las más adecuadas a las condiciones de funcionamiento de La Casona 23yB.
- Hacer una arquitectura, de manera que permita desarrollar el sistema que gestione el funcionamiento de La Casona 23yB de la mejor forma posible.
- Propiciar eficiencia en el funcionamiento del sistema de La Casona 23yB.
- Seleccionar las herramientas idóneas para llevar a cabo el proyecto y elegir la plataforma en la que se desarrollará este.

Para el desarrollo de una solución a los problemas planteados, se especifica como preguntas de investigación:

- ¿Cuáles son los principales subsistemas que conformarían el sistema de La Casona 23yB?
- ¿Cuál es el tipo de arquitectura de software a emplear para el desarrollo del sistema de La Casona 23yB?
- ¿Permite la arquitectura seleccionada implementar la funcionalidad básica deseada?
- ¿Responde la arquitectura seleccionada a los atributos de calidad establecidos por los clientes?

Para cumplir con estos objetivos se plantean un grupo de **Tareas de la investigación** que permitan realizar una implementación satisfactoria del sistema, las siguientes son:

- Buscar información sobre los distintos tipos de arquitecturas.
- Realizar plan de captura de requisitos del Proyecto 23yB.
- Selección y revisión bibliográfica sobre los distintos enfoques existentes de la arquitectura en proyectos de informatización.
- Evaluar el contenido de la información obtenida sobre el funcionamiento de proyectos informatizados que se investigan.
- Indagación sobre una propuesta de arquitectura en la que tiene que funcionar el sistema, su arquitectura hardware y darse cumplimiento a los requisitos no funcionales, todo conforme con las condiciones y características del Proyecto 23yB
- Implementar la arquitectura del Proyecto 23yB de manera tal que satisfaga las necesidades del software.

CAPÍTULO 1.FUNDAMENTACIÓN TEÓRICA

Proyecto 23yB

La Casona 23yB destinada a profesores de Universidad para Todos y Canales Educativos, que es un personal dedicado a la teleformación educativa en Cuba desde hace más de 5 años, será el espacio propicio para reunir a todos los profesionales que han colaborado con estos grandes proyectos de La Batalla de Ideas. Su existencia es de vital importancia en el fomento de la cultura y la instrucción del pueblo cubano. Se ha proyectado como una organización moderna, que brinda servicios de excelencia, con personal altamente calificado y tecnología de avanzada. El logro de estos objetivos estará condicionado por los valores que promoverá esta entidad, en la misma se va a instalar una infraestructura tecnológica y física, para así brindar servicios de localización, búsqueda, gestión y utilización de información necesaria para desarrollar las clases y actividades de los programas de Universidad para Todos y los Canales Educativos, además permitirá acceder a sistemas de capacitación y formación permanente de estos profesionales en el uso de las nuevas tecnologías de la información y las comunicaciones. Esta institución ha de convertirse en el lugar donde dichos profesionales se reúnan y preparen los materiales necesarios para impartir diferentes cursos, organizar eventos; además de que puedan contar con una serie de recursos y servicios que posibiliten el apoyo necesario para desarrollar exitosamente todas las actividades que se provean. Se debe lograr que la institución aporte a la gestión del conocimiento de estos profesionales y de proyectos tan importantes para el desarrollo intelectual del país como lo son Universidad para Todos y los cursos que se imparten en los Canales Educativos. Debe reinar el esfuerzo a satisfacer las necesidades de estos profesionales, quienes se convertirán en usuarios de la nueva institución. En la institución se proyectarán y diseñarán servicios generales que puedan brindarse a todos los profesionales que asistan a la misma, hasta tanto se apliquen estudios de necesidades de usuarios que posibiliten una especialización tal de los servicios que responda a las necesidades reales del personal que hace uso de los mismos. El contacto que se tenga con los profesionales de los proyectos, posibilitará conocer la manera en que ellos desean recibirlos, lo cual contribuirá a la personalización de dichos servicios. De forma general la misión de La Casona 23yB es que va a ser una institución coordinadora de las potencialidades institucionales y profesionales creadas por la

revolución en función de elevar la calidad de la educación y la cultura general integral de la población cubana.

Diseño del sistema de información para La Casona 23 y B

Los cambios en los modelos de enseñanza han transformado el escenario donde se insertan las bibliotecas como sistemas, en este ámbito surge la necesidad de crear un modelo de organización de información que intente generar nuevos espacios y funciones, aprovechando los procesos tradicionales de las bibliotecas e integrando recursos tecnológicos, audiovisuales, instalaciones y medios para la edición y creación de materiales interactivos y/o electrónicos que den soporte a las necesidades de aprendizaje.

El diseño del sistema de información se genera con el objetivo de que los profesores de Universidad para Todos y los Canales Educativos puedan: localizar recursos de información para la preparación de sus clases, acceder a servicios de formación y aprendizaje que permitan crear habilidades para una mejor selección de la información a utilizar en la preparación de clases y en su desempeño como tele profesor, utilizar sistemas de búsqueda y recuperación de información que brinde un alto nivel de recobrado en relación al volumen de información disponible, acceder a fuentes de información que propicien un nivel de intercambio y análisis.

El diseño de los servicios de información debe responder a las necesidades de la comunidad de usuarios de La Casona 23yB. Se deben tomar en consideración temáticas que respondan a sus necesidades, aspectos culturales, medios de transmisión. Los profesores de Universidad para Todos y los Canales Educativos conforman, potencialmente, el universo de usuarios de la institución. Esta nueva institución no permite la realización de un estudio completo del universo de usuarios, por lo que se tomarán en cuenta las necesidades de un segmento de los profesores, además del personal interno de la entidad.

En la institución se brindarán servicios de consulta de materiales audiovisuales y multimedia al igual que otros materiales digitales e impresos, así como préstamos de dichos materiales. Entre estos servicios se encuentran: Servicios de localización, búsqueda, y gestión de la información necesaria para el desarrollo de los programas por la vía tradicional, audiovisual, formato electrónico, en línea, entre otros, observatorio y monitoreo de canales satelitales, Internet y agencias de información, con el propósito de su utilización por los profesores en Universidad Para Todos, los Canales Educativos y para su superación, capacitación y superación permanente del claustro de Universidad Para Todos, los Canales Educativos y otros profesionales vinculados a los programas de la revolución según sus demandas o necesidades

detectadas, en las modalidades presencial, semipresencial o a distancia, servicios telemáticos necesarios para el acceso a la información y el conocimiento que demanden estos programas, investigaciones sobre temáticas afines al desarrollo de estos programas, velar por la protección de los productos que se generan como resultado del desarrollo de estos programas de la revolución, digitalización y reproducción de documentos, grabación, digitalización y reproducción de programas audiovisuales, reservación de locales y equipos.

Arquitectura del Software

Lo primero que debemos hacer es conocer todo lo referente sobre la Arquitectura de Software (ó AS) con el propósito puntual de brindar una visión de conjunto lo más estructurada posible, para luego establecer el papel de esta disciplina emergente en el proyecto.

Breve Historia de la Arquitectura de Software

Todavía no se ha escrito una historia aceptable de la AS. Desde que Mary Shaw o David Garlan reseñaran escuetamente la prehistoria de la especialidad a principios de los 90, los mismos párrafos han sido reutilizados una y otra vez en la literatura, sin mayor exploración de las fuentes referidas en la reseña primaria y con prisa por ir al grano, que usualmente no es de carácter histórico. En este estudio se ha optado, más bien, por inspeccionar las fuentes más de cerca, con el objeto de definir con mayor claridad el contexto, entender que muchas contribuciones que pasaron por complementarias han sido en realidad antagónicas y comprender mejor por qué algunas ideas que surgieron hace cuatro décadas demoraron un cuarto de siglo en materializarse.

Situar las inflexiones de la breve historia de la AS en un contexto temporal, asimismo, ayudará a comprender mejor cuáles son sus contribuciones perdurables y cuáles sus manifestaciones contingentes al espíritu de los tiempos y a las modas tecnológicas que se han ido sucediendo.

Si bien la AS acostumbra remontar sus antecedentes al menos hasta la década de 1960, su historia no ha sido tan continua como la del campo más amplio en el que se inscribe, la ingeniería de software [Pfl02]. Después de las tempranas inspiraciones del legendario Edsger Dijkstra, de David Parnas y de Fred Brooks, la AS quedó en estado de vida latente durante unos cuantos años, hasta comenzar su expansión explosiva con los manifiestos de Dewayne Perry de AT&T Bell Laboratories de New Jersey y Alexander Wolf de la Universidad de Colorado [PW92]. 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 entonces de una práctica joven, 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, en la misma universidad. Comencemos entonces por el principio, aunque siempre cabrá la posibilidad de discutir cuál puede haber sido el momento preciso en el que todo comenzó.

Cada vez que se narra la historia de la arquitectura de software (o de la ingeniería de software, según el caso), se reconoce que en un principio, hacia 1968, Edsger Dijkstra, de La Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [Dij68a]. Dijkstra, 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 lo que luego expresarían Niklaus Wirth [Wir71] como stepwise refinement y DeRemer y Kron [DK76] como programming-in-the large (o programación en grande), ideas que poco a poco irían decantando entre los ingenieros primero y los arquitectos después.

En la conferencia de la NATO de 1969, un año después de la sesión en que se fundara la ingeniería de software, P. I. Sharp formuló estas sorprendentes apreciaciones comentando las ideas de Dijkstra:

“Pienso que tenemos algo, aparte de la ingeniería de software: algo de lo que hemos hablado muy poco pero que deberíamos poner sobre el tapete y concentrar la atención en ello. Es la cuestión de la arquitectura de software. La arquitectura es diferente de la ingeniería. Como ejemplo de lo que quiero decir, echemos una mirada a OS/360. Partes de OS/360 están extremadamente bien codificadas. Partes de OS, si vamos al detalle, han utilizado técnicas que hemos acordado constituyen buena práctica de programación. La razón de que OS sea un amontonamiento amorfo de programas es que no tuvo arquitecto. Su diseño fue delegado a series de grupos de ingenieros, cada uno de los cuales inventó su propia arquitectura. Y cuando esos pedazos se clavaron todos juntos no produjeron una tersa y bella pieza de software [NATO76].”

Sharp continúa su alegación afirmando que con el tiempo probablemente llegue a hablarse de “la escuela de arquitectura de software de Dijkstra” y se lamenta que en la industria de su tiempo se preste tan poca o ninguna atención a la arquitectura. La frase siguiente también es extremadamente visionaria:

“Lo que sucede es que las especificaciones de software se consideran especificaciones funcionales. Sólo hablamos sobre lo que queremos que haga el programa. Es mi creencia que cualquiera que sea responsable de la implementación de una pieza de software debe especificar más que esto. Debe especificar el diseño, la forma; y dentro de ese marco de referencia, los programadores e ingenieros deben crear algo. Ningún ingeniero o programador, ninguna herramienta de programación, nos ayudará, o ayudará al negocio del software, a maquillar un diseño feo. El control, la administración, la educación y todas las cosas buenas de las que hablamos son importantes; pero la gente que implementa debe entender lo que el arquitecto tiene en mente.”

Nadie volvió a hablar del asunto en esa conferencia, sin embargo. Por unos años, “arquitectura” fue una metáfora de la que se echó mano cada tanto, pero sin precisión semántica ni consistencia pragmática. En 1969 Fred Brooks Jr y Ken Iverson llamaban arquitectura a la estructura conceptual de un sistema en la perspectiva del programador. En 1971, C. R. Spooner tituló uno de sus ensayos “Una arquitectura de software para los 70s” [Spo71], sin que la mayor parte de la historiografía de la AS registrara ese antecedente.

En 1975, Brooks, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa [Bro75], empleando una nomenclatura que ya nadie aplica de ese modo. En el mismo texto, identificaba y razonaba sobre las estructuras de alto nivel y reconocía la importancia de las decisiones tomadas a ese nivel de diseño. También distinguía entre arquitectura e implementación; mientras aquella decía qué hacer, la implementación se ocupa de cómo. Aunque el concepto de AS actual y el de Brooks difieren en no escasa medida, el texto de Brooks The mythical man-month sigue siendo, un cuarto de siglo más tarde, el más leído en ingeniería de software.

En la década de 1970, otro precursor importante, David Parnas, 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.

En 1972, Parnas publicó un ensayo en el que discutía la forma en que la modularidad en el diseño de sistemas podía mejorar la flexibilidad y el control conceptual del sistema, acortando los tiempos de desarrollo [Par72]. Introdujo entonces el concepto de ocultamiento de información (information hiding), uno de los principios de diseño fundamentales en diseño de software aún en la actualidad. La herencia de este concepto en la ingeniería y la arquitectura ulterior es inmensa, y se confunde estrechamente con la idea de abstracción.

El pensamiento de Parnas sobre familias de programas, en particular, anticipa ideas que luego habrían de desarrollarse a propósito de los estilos de arquitectura:

En la década de 1980, los métodos de desarrollo estructurado demostraron no escalar suficientemente y fueron dejando el lugar a un nuevo paradigma, el de la programación orientada a objetos. Paralelamente, hacia fines de la década de 1980 y comienzos de la siguiente, la expresión arquitectura de software comienza a aparecer en la literatura para hacer referencia a la configuración morfológica de una aplicación.

El primer estudio en que aparece la expresión “arquitectura de software” en el sentido en que hoy lo conocemos es sin duda el de Perry y Wolf [PW92]; ocurrió tan tarde como en 1992, aunque el trabajo se fue gestando desde 1989. En él, los autores proponen concebir la AS por analogía con la arquitectura de edificios, una analogía de la que luego algunos abusaron, otros encontraron útil y para unos pocos ha devenido inaceptable [BR01].

Dando cumplimiento a las profecías de Perry y Wolf, la década de 1990 fue sin duda la de la consolidación y diseminación de la AS en una escala sin precedentes. Las contribuciones más importantes surgieron en torno del instituto de ingeniería de la información de la Universidad Carnegie Mellon (CMU SEI). En la misma década, demasiado pródiga en acontecimientos, surge también la programación basada en componentes, que en su momento de mayor impacto impulsó a algunos arquitectos mayores, como Paul Clements [Cle96b], a afirmar que la AS promovía un modelo que debía ser más de integración de componentes preprogramados que de programación.

Un segundo gran tema de la época fue el surgimiento de los patrones, cristalizada en dos textos fundamentales, el de la Banda de los Cuatro en 1995 [Gof95] y la serie POSA desde 1996 [BMR+96]. El primero de ellos promueve una expansión de la programación orientada a objetos, mientras que el segundo desenvuelve un marco ligeramente más ligado a la AS. El originador de la idea de patrones fue Christopher Alexander, quien incidentalmente fue arquitecto de edificios.

Como quiera que sea, la AS de este período realizó su trabajo de homogeneización de la terminología, desarrolló la tipificación de los estilos arquitectónicos y elaboró lenguajes de descripción de arquitectura (ADLs), temas que en este estudio se tratan en documentos separados.

Uno de los acontecimientos arquitectónicos más importantes del año 2000 fue la hoy célebre tesis de Roy Fielding que presentó el modelo REST, el cual establece definitivamente el tema de las tecnologías de Internet y los modelos orientados a servicios y recursos en el centro de las preocupaciones de la disciplina [Fie00]. En el mismo año se publica la versión definitiva de la recomendación IEEE Std 1471, que procura homogeneizar y ordenar la nomenclatura de descripción arquitectónica y homologa los estilos como un modelo fundamental de representación conceptual.

En el siglo XXI, la AS aparece dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación, diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos. Todo lo que se ha hecho en ingeniería debe formularse de nuevo, integrando la AS en el conjunto.

La semblanza que se ha trazado no es más que una visión selectiva de las etapas recorridas por la AS. Los lineamientos de ese proceso podrían dibujarse de maneras distintas, ya sea enfatizando los hallazgos formales, las intuiciones dominantes de cada período o las diferencias que median entre la abstracción cualitativa de la arquitectura y las cuantificaciones que han sido la norma en ingeniería de software.

Diferencias entre Arquitectura y Diseño

Una vez que se reconoce la diferencia, que nunca debió ser menos que obvia, entre diseño e implementación, o entre vistas conceptuales y vistas tecnológicas ¿Es la AS solamente otra palabra para designar el diseño? Como suele suceder, no hay una sola respuesta, y las que hay no son específicas. La comunidad de AS, en particular la de procedencia académica, sostiene que ésta difiere sustancialmente del mero diseño. Pero Taylor y Medvidovic, por ejemplo, señalan que la literatura actual mantiene en un estado ambiguo la relación entre ambos campos, albergando diferentes interpretaciones y posturas:

- Una postura afirma que arquitectura y diseño son lo mismo.
- Otra, en cambio, alega que la arquitectura se encuentra en un nivel de abstracción por encima del diseño, o es simplemente otro paso (un artefacto) en el proceso de desarrollo de software.

- Una tercera establece que la arquitectura es algo nuevo y en alguna medida diferente del diseño (pero de qué manera y en qué medida se dejan sin especificar).

Taylor y Medvidovic estiman que la segunda interpretación es la que se encuentra más cerca de la verdad. En alguna medida, la arquitectura y el diseño sirven al mismo propósito. Sin embargo, el foco de la AS en la estructura del sistema y en las interconexiones la distingue del diseño de software tradicional, tales como el diseño orientado a objetos, que se concentra más en el modelado de abstracciones de más bajo nivel, tales como algoritmos y tipos de datos. A medida que la arquitectura de alto nivel se refina, sus conectores pueden perder prominencia, distribuyéndose a través de los elementos arquitectónicos de más bajo nivel, resultando en la transformación de la arquitectura en diseño.

Stephen Albin se pregunta en qué difiere la AS de las metodologías de diseño bien conocidas como la orientación a objetos. Entonces plantea que la AS es una metáfora relativamente nueva en materia de diseño de software y en realidad abarca también las metodologías de diseño, así como metodologías de análisis. El arquitecto de software contemporáneo, plantea Albin, ejecuta una combinación de roles como los de analista de sistemas, diseñador de sistemas e ingeniero de software. Pero la arquitectura es más que una recolocación de funciones. Esas funciones pueden seguir siendo ejecutadas por otros, pero ahora caen comúnmente bajo la orquestación del jefe de arquitectos. El concepto de arquitectura intenta subsumir las actividades de análisis y diseño en un framework de diseño más amplio y más coherente. Las organizaciones se están dando cuenta que el alto costo del desarrollo de software requiere ser sometido a algún control y que muchas de las ventajas prometidas por las metodologías aún no se han materializado. Pero la arquitectura es algo más integrado que la suma del análisis por un lado y el diseño por el otro. La integración de metodologías y modelos, concluye Albin, es lo que distingue la AS de la simple yuxtaposición de técnicas de análisis y de diseño.

Para Shaw y Garlan la AS es el primer paso en la producción de un diseño de software, en una secuencia que distingue tres pasos:

1) Arquitectura. Asocia las capacidades del sistema especificadas en el requerimiento con los componentes del sistema que habrán de implementarla. La descripción arquitectónica incluye componentes y conectores (en términos de estilos) y la definición de operadores que crean sistemas a partir de subsistemas o, en otros términos, componen estilos complejos a partir de estilos simples.

2) Diseño del código. Comprende algoritmos y estructuras de datos; los componentes son aquí primitivos del lenguaje de programación, tales como números, caracteres, punteros e hilos de control. También hay operadores primitivos.
3) Diseño ejecutable. Remite al diseño de código a un nivel de detalle todavía más bajo y trata cuestiones tales como la asignación de memoria, los formatos de datos, etcétera.

En opinión de Clements el diseño basado en arquitectura representa un paradigma de desarrollo que difiere de maneras fundamentales de las alternativas conocidas actualmente. En muchos sentidos, es diferente del diseño orientado a objetos (OOD) en la misma medida en que éste difería de sus predecesores. La AS deberá nutrir una comunidad de practicantes estableciendo una cultura en la que las ideas arquitectónicas puedan florecer. Una cuestión técnica que deberá abordarse es la creación de una articulación precisa de un paradigma de diseño basado en arquitectura (o posiblemente más de uno) y las cuestiones de proceso asociadas.
En una presentación de 1997, Dewayne Perry, uno de los fundadores de la disciplina, bosquejó la diferencia entre arquitectura y diseño. La arquitectura, una vez más (todo el mundo insiste en ello) concierne a un nivel de abstracción más elevado; se ocupa de componentes y no de procedimientos; de las interacciones entre esos componentes y no de las interfaces; de las restricciones a ejercer sobre los componentes y las interacciones y no de los algoritmos, los procedimientos y los tipos. En cuanto a la composición, la de la arquitectura es de grano grueso, la del diseño es de fina composición procedural; las interacciones entre componentes en arquitectura tienen que ver con un protocolo de alto nivel (en el sentido no técnico de la palabra), mientras que las del diseño conciernen a interacciones de tipo procedural (rpc, mensajes, llamadas a rutinas).
En los primeros años del nuevo siglo, la AS precisó la naturaleza del proceso de diseño como metodología en diversos modelos de diseño basados en arquitectura o ABD. Esta metodología considera que el diseño arquitectónico es el de más elevado nivel de abstracción, pero debe hacer frente al hecho de un requerimiento todavía difuso y al hecho de que, en ese plano, las decisiones que se tomen serán las más críticas y las más difíciles de modificar. Fundándose en el concepto de arquitectura conceptual de Hofmeister, Nord y Soni y en un modelo de vistas, similar al 4+1 o a las vistas del modelo arquitectónico de Microsoft, el ABD describe el sistema en función de los principales elementos y las relaciones entre ellos. El proceso se basa en tres fundamentos: (1) la descomposición de la función (usando técnicas bien

establecidas de acoplamiento y cohesión), (2) la realización de los requerimientos de calidad y negocios a través de los estilos arquitectónicos, y (3) las plantillas de software, un concepto nuevo que incluye patrones que describen la forma en que todos los elementos de un tipo interactúan con los servicios compartidos y la infraestructura

Conceptos fundamentales

Más allá de que hoy existan numerosos conceptos en el plano detallado de las técnicas y metodologías, la AS se articula alrededor de unos pocos conceptos y principios esenciales y unas pocas herramientas características.

Estilos

En el texto fundacional de la AS, 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.

Concisamente descriptos, los estilos conjugan elementos (o “componentes”, como se los llama aquí), conectores, configuraciones y restricciones. Al estipular los conectores como elemento de juicio de primera clase, el concepto de estilo, incidentalmente, se sitúa en un orden de discurso y de método que el modelado orientado a objetos en general y UML en particular no cubren satisfactoriamente. La descripción de un estilo se puede formular en lenguaje natural o en diagramas, pero lo mejor es hacerlo en un lenguaje de descripción arquitectónica o en lenguajes formales de especificación.

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. Es digno de señalarse el empeño por subsumir todas las formas existentes de aplicaciones en un conjunto de dimensiones tan modestas. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos. Algunos estilos típicos son las arquitecturas basadas en flujo de datos, las peer-to-peer, las de invocación implícita, las jerárquicas, las centradas en datos o las de intérprete-máquina virtual.

Patrones Arquitectónicos de Usabilidad

El concepto de patrón más ampliamente utilizado en el desarrollo del software es el patrón de diseño, que se utiliza particularmente en el paradigma orientado a objetos. En este contexto, un patrón de diseño es una descripción de las clases y de los objetos que trabajan conjuntamente para resolver un problema concreto. Estos patrones muestran una solución a un problema, que se ha obtenido a partir de su uso en otras aplicaciones diferentes. No obstante, debe señalarse que un patrón de diseño no debe verse como una solución única u original, sino como una posible solución.

Dado que el patrón de usabilidad se ha definido como un mecanismo a ser incorporado en el diseño de una arquitectura software a fin de abordar una propiedad de usabilidad concreta, un patrón arquitectónico determinará cómo se incorporará este patrón de usabilidad en una arquitectura software; es decir, qué efecto tendrá la inclusión del patrón de usabilidad en los componentes de la arquitectura del sistema. Abstrayendo la definición del patrón de diseño, un patrón arquitectónico puede definirse como una descripción de los componentes de un diseño y de la comunicación entre ellos para proporcionar una solución a un patrón de usabilidad. Al igual que los patrones de diseño, los patrones arquitectónicos reflejarán una posible solución a un problema: la incorporación de un patrón de usabilidad concreto en un diseño software.

Por lo tanto, el patrón arquitectónico es el último eslabón en la cadena atributo, propiedad y patrón de usabilidad, y conecta la usabilidad del sistema software con la arquitectura de éste.

Lenguajes de Definición de Arquitecturas

Uno de los roles de la arquitectura de software es el de servir como conducto de comunicación y así facilitar el entendimiento del sistema y la abstracción de alto nivel. Por lo tanto los ADLs deben ser capaces de modelar información estructural con una sintaxis comprensible.

Un lenguaje de definición de arquitecturas (ADL) sirve para describir una arquitectura de software, este debe hacerse cargo de todas estas características:

Según Medvidovic, existen ciertas características que son deseables en un ADL; éstas se resumen en la tabla siguiente y se detallan seguidamente. Los elementos indicados (componentes, conectores, configuraciones e interfaces de componentes) son los mínimos necesarios para que un lenguaje de especificación pueda ser considerado un ADL.

Elemento	Características
Componentes	Interfaces Tipos Semántica Evolución Propiedades no funcionales
Conectores	Interfaces Tipos Semántica Evolución Propiedades no funcionales
Configuraciones	Comprensibilidad Composición jerárquica Refinamiento y seguimiento Heterogeneidad Escalabilidad Evolucionabilidad Dinamismo Restricciones Propiedades no funcionales

Tabla1: Características de modelamiento requeridas para un ADL.

Para realizar la descripción de la arquitectura los ADLs utilizan esencialmente componentes que son las piezas físicas de la implementación de un sistema, y los conectores para modelar la interacción. Estos conectores pueden ser implícitos en algunos ADLs o bien modelarse como elementos de primera clase, dependiendo del ADL y del patrón de arquitectura que se esté modelando.

Las componentes son las entidades computacionales activas de un sistema. Ellas realizan tareas mediante cómputo interno y comunicación externa con el resto del sistema. La relación entre una componente y su entorno se define explícitamente como una colección de puntos de interacción o puertos. Una componente tiene sus propios datos y espacio de ejecución independientes, aunque podría compartirlos con otras componentes. El componente debe tener diferentes características como son:

Interfaces. Es el conjunto de puntos de interacción entre una componente y su entorno. La interfaz especifica los servicios que provee una componente (operaciones, mensajes y variables) y también los servicios que la componente requiere de otras componentes del sistema.

Tipos. Los tipos de una componente son abstracciones que encapsulan funcionalidades en bloques reutilizables. Un tipo de componente puede ser instanciado múltiples veces en una sola arquitectura o ser reutilizado en diferentes arquitecturas.

Semántica. Se define la semántica como un modelo de alto nivel del comportamiento de la componente. Este modelo es necesario para realizar análisis, reforzar restricciones arquitecturales, y para asegurar la consistencia entre diferentes niveles de abstracción.

Evolución. Los ADLs pueden permitir la evolución de componentes definiendo tipos de componentes y permitiendo el refinamiento mediante subtipado (subtyping).

Propiedades no funcionales. Estas propiedades no tienen relación con la funcionalidad sino con características de calidad de la componente tales como seguridad, rendimiento y portabilidad.

Los conectores definen la interacción entre los componentes. Cada conector provee de una forma para que una colección de puertos esté en contacto y define lógicamente el protocolo a través del cual un conjunto de componentes puede interactuar. Los puertos definen los puntos de interacción de los conectores. Al igual que las componentes, un conector tiene una interfaz, la cual consiste en un conjunto de roles. Los conectores tienen las siguientes características.

Interfaz. Es el conjunto de puntos de interacción entre un conector y una componente u otro conector. Las interfaces de conectores permiten una conectividad apropiada entre componentes y su interacción en una arquitectura.

Tipos. Tipos de conectores son abstracciones que encapsulan la comunicación, coordinación y decisiones de mediación de componentes. Interacciones de nivel de arquitecturas son caracterizadas por protocolos complejos identificados para diferentes patrones de arquitectura

Semántica. La semántica de conectores se define como un modelo de alto nivel del comportamiento de un conector. La semántica expresa funcionalidad a nivel de la aplicación y la especificación de protocolos de interacción.

Evolución. La evolución de un conector se refiere a la modificación de las propiedades de un conector, o sea, de la modificación de su interfaz, semántica, o restricciones entre las dos.

Propiedades no funcionales. Las propiedades no funcionales de los conectores no se pueden derivar completamente de la especificación de su semántica. Representan requerimientos para implementar correctamente conectores. Permiten simulaciones de comportamiento en tiempo de ejecución, reforzamiento de restricciones, etc.

Una configuración o topología es una colección de instancias de componentes que interactúan mediante instancias de conectores. En otras palabras, una topología es un grafo de componentes y conectores conectados que describen la estructura de la arquitectura. Las características de nivel de configuración se agrupan en tres categorías generales: calidad de la descripción de la configuración, calidad de la descripción del sistema, propiedades de la descripción del sistema. Las configuraciones tienen las siguientes características.

Especificaciones entendibles. Uno de los roles de la arquitectura de software es el de servir como conducto de comunicación y así facilitar el entendimiento del sistema y la abstracción de alto nivel.

Composición jerárquica. Es un mecanismo que permite que una arquitectura sea definida con distintos niveles de detalle.

Refinamiento y seguimiento. Los ADLs deben proveer refinamiento de arquitecturas en sistemas ejecutables y seguimiento de los cambios a través de los distintos niveles.

Heterogeneidad. Las arquitecturas de software deben facilitar el desarrollo de sistemas de gran escala mediante la existencia de componentes y conectores de distinta granularidad, posiblemente especificadas en diferentes lenguajes de modelamiento e implementadas en diferentes lenguajes de programación.

Escalabilidad. Los ADLs deben soportar la especificación y el desarrollo de sistemas de gran escala que pueden crecer en el futuro.

Evolucionabilidad. Una arquitectura evoluciona para reflejar y permitir la evolución de familias de sistemas de software.

Dinamismo. Se refiere a modificar la arquitectura y reflejar esas modificaciones en el sistema mientras éste se ejecuta.

Propiedades no funcionales. Algunas de las propiedades no funcionales están a nivel del sistema más que a nivel de componentes y conectores individuales. Propiedades no funcionales a nivel de sistema son necesarias para seleccionar apropiadamente componentes y conectores, realizar análisis y reforzar restricciones, entre otras.

Las restricciones están presentes en cada una de las tres características anteriormente expuestas (componentes, conectores y configuración). Las restricciones son propiedades o afirmaciones sobre un sistema o alguna de sus partes. Entonces a continuación se mostrarán las restricciones para cada una de las características mencionadas.

Restricciones en componentes. Una componente puede ser restringida mediante atributos, restringiendo el número de asociaciones que puede tener un puerto, o indicando atributos no funcionales, tiempo de ejecución o deadlines.

Restricciones en conectores. Las restricciones de los conectores aseguran la adherencia a protocolos de interacción. Además establecen dependencias entre conectores y refuerzan el uso de límites.

Restricciones en configuración. Restricciones que representan dependencias en una configuración complementan aquellas específicas a las componentes y conectores.

Definiciones

La Arquitectura del Software en una disciplina que se encuentra actualmente en desarrollo, por lo que no hay una definición formal de esta, la cual sirva como base para cualquier estudio que se haga de esta disciplina, por este motivo se ofrece a continuación un grupo de definiciones.

Una definición reconocida es la de Clements: La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

A despecho de la abundancia de definiciones del campo de la AS, existe en general acuerdo de que ella se refiere a la estructura a grandes rasgos del sistema, estructura consistente en componentes y relaciones entre ellos. Estas cuestiones estructurales se vinculan con el diseño, pues la AS es después de todo una forma de diseño de software que se manifiesta tempranamente en el proceso de creación de un sistema; pero este diseño ocurre a un nivel más abstracto que el de los algoritmos y las estructuras de datos. En el que muchos consideran un ensayo seminal de la disciplina, Mary Shaw y David Garlan sugieren que dichas cuestiones estructurales incluyen organización a grandes rasgos y estructura global de control; protocolos para la comunicación, la sincronización y el acceso a datos; la asignación de funcionalidad a elementos del diseño; la distribución física; la composición de los elementos de diseño; escalabilidad y rendimiento; y selección entre alternativas de diseño.

Ante el número y variedad de definiciones existentes de AS, Mary Shaw y David Garlan proporcionaron una sistematización iluminadora, explicando las diferencias entre definiciones en función de distintas clases de modelos. Destilando las definiciones y los puntos de vista implícitos o explícitos, los autores clasifican los modelos de esta forma:

Modelos estructurales: Sostienen que la AS está compuesta por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizado por el desarrollo de lenguajes de descripción arquitectónica (ADLs).

Modelos de framework: Son similares a la vista estructural, pero su énfasis primario radica en la (usualmente una sola) estructura coherente del sistema completo, en vez de concentrarse en su composición. Los modelos de framework a menudo se refieren a dominios o clases de problemas específicos. El trabajo que ejemplifica esta variante incluye arquitecturas de software específicas de dominios, como CORBA, o modelos basados en CORBA, o repositorios de componentes específicos, como PRISM.

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.

Modelos de proceso: Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción. En esta perspectiva, la arquitectura es el resultado de seguir un argumento (script) de proceso. Esta vista se ejemplifica con el actual trabajo sobre programación de procesos para derivar arquitecturas.

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 framework particular.

Ninguna de estas vistas excluye a las otras, ni representa un conflicto fundamental sobre lo que es o debe ser la AS. Por el contrario, representan un espectro en la comunidad de investigación sobre distintos énfasis que pueden aplicarse a la arquitectura: sobre sus partes constituyentes, su totalidad, la forma en que se comporta una vez construida, o el proceso de su construcción. Tomadas en su conjunto, destacan más bien un consenso.

Ciclo de vida de la arquitectura, su importancia.

Las necesidades actuales que tiene toda organización para el logro de sus objetivos, demandan la construcción de grandes y complejos sistemas de software que requieren de la combinación de diferentes tecnologías y plataformas de hardware y software para alcanzar un funcionamiento acorde con dichas necesidades. Lo anterior, exige de los profesionales dedicados al desarrollo de software poner especial atención y cuidado al diseño de la arquitectura, bajo la cual estará soportado el funcionamiento de sus sistemas.

Si una arquitectura de software se encuentra deficiente en su concepto o diseño, o en el peor de lo casos, no contamos con la del sistema que desarrollamos, tendremos grandes posibilidades de construir un sistema que no alcanzará el total de los requerimientos establecidos. Esto, indudablemente, nos generará un re-trabajo complicado o, peor aún, nos podrá llevar al fracaso del sistema de software cuando se encuentre en operación.

De esta manera, es necesario conocer y comprender los elementos que deben atacarse al diseñar una arquitectura de software, entendida como un término que se ha venido perfilando en los últimos años por los profesionales de la industria, a pesar de que es un tópico que ha sido desarrollado por los expertos del campo de la ingeniería de software desde hace muchos años atrás. Son pocos los profesionales que conocen lo que en realidad abarca este tema y cómo debe diseñarse la arquitectura de un sistema de software, lo cual se debe al desconocimiento generalizado de esta importante etapa del ciclo de vida de un sistema. Regularmente, se pasa de la especificación de requerimientos a un diseño somero y a la codificación del sistema.

El diseño de software se divide en dos partes importantes: diseño arquitectónico (también conocido como diseño de alto nivel) y diseño detallado. El diseño de la arquitectura de software ocurre inmediatamente después de la especificación de los requerimientos de software y considera como elementos principales los siguientes: componentes de software, propiedades de dichos componentes y la comunicación entre ellos. El diseño detallado se lleva a cabo justo antes de la codificación, y forma parte de las primeras tareas del desarrollador; describe la lógica, el control jerárquico, estructura de datos, empacado de componentes, etcétera.

El desarrollo de la arquitectura de software es una de las etapas fundamentales y, en muchos casos, la más importante en el desarrollo de software, pues es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que esperan del sistema.

Desarrollar una arquitectura de software es como llevar a cabo el diseño arquitectónico de un edificio que será construido. Para construir, los ingenieros, albañiles, plomeros, electricistas, etc. requieren estudiar y comprender los planos de los cimientos, la estructura, y de toda la infraestructura necesaria para que cuente con servicios tales como: luz, agua, teléfono, red de datos, etc. Cuando no existen estos planos que nos guíen, estaremos construyendo algo que se nos viene a la imaginación justo en el momento de

realizarlo. Por ello, también es importante que los diferentes interesados en el sistema se involucren en el diseño de la arquitectura pues, con ello, se podrá acordar y consensuar de una mejor manera la solución a la que se llegue después de conocer los requerimientos.

Como definición formal tomaremos la siguiente: “la arquitectura de software de un sistema es la estructura o estructuras del sistema, lo cual abarca componentes de software, las propiedades visibles externamente de esos componentes, y las relaciones entre ellas”. De esta manera, la arquitectura de software permite representar de forma concreta la estructura y funcionamiento interno de un sistema.

Para comenzar el desarrollo de la arquitectura de software es necesario partir de un documento de especificación de requerimientos, en caso contrario, deberemos trabajar de manera formal en una etapa de requerimientos para definir de manera detallada lo que se espera del sistema. El documento debe contener requerimientos funcionales (del negocio, de usuario, de sistema, entre otros) y requerimientos no funcionales (reglas de negocio, atributos de calidad del sistema, interfaces externas y políticas, por mencionar algunas).

Un elemento crítico y muy importante que se debe considerar en una arquitectura de software, y en lo que precisamente está basado el diseño, son los requerimientos no funcionales del sistema, específicamente los atributos de calidad establecidos para el mismo, es decir, atributos como: desempeño, confiabilidad, seguridad, facilidad de modificación, facilidad de uso, robustez, portabilidad, escalabilidad, reutilización, disponibilidad, etcétera.

Ahora bien, resulta imposible aceptar todos los atributos de calidad para un sistema; cada interesado en dicho sistema (usuario, cliente, desarrollador, arquitecto, patrocinador, etc.) estará preocupado por alguno en específico, pero es imposible alcanzar todos, pues es sabido que alcanzar ciertos atributos de calidad impide que se logren otros, y es cuando se dice que entran en conflicto. El secreto está en priorizarlos y en determinar cuáles son los verdaderamente importantes para el cliente y cuáles está dispuesto a sacrificar para alcanzar su objetivo.

De no considerar lo anterior cometeremos un error grave, pues ningún sistema puede cumplir con todos los atributos de calidad al 100%, resultaría incoherente por el conflicto que puede existir entre ellos. Es por esto que, en muchas ocasiones, los usuarios y clientes quedan insatisfechos con un sistema, pues esperaban todos los atributos de calidad que acordaron con el proveedor. Muy probablemente el analista y arquitecto, por desconocimiento, aceptaron y se comprometieron a cumplir con todos ellos sin analizarlos.

En ese sentido, podemos determinar que la arquitectura de software se encuentra influenciada por los involucrados o interesados en el desarrollo del sistema de software, la organización para la que está siendo desarrollado, los requerimientos no funcionales, el ambiente técnico y la experiencia del arquitecto.

Un vez generada y documentada la arquitectura de software, ésta debe evaluarse para verificar que cumpla con todos los requerimientos; específicamente con los atributos de calidad establecidos. Dicha evaluación puede realizarse mediante técnicas cualitativas, como cuestionarios o escenarios, o a través de técnicas cuantitativas, como simulaciones o modelos matemáticos. En la literatura existen diferentes métodos de evaluación para verificar desde múltiples atributos de calidad hasta algunos en específico. Ejemplos de estos métodos de evaluación son ATAM, ABAS, SAAM, SNA, ALMA, RMA, teoría de colas, teoría de confiabilidad, entre otras.

De manera general, podemos decir que las tareas realizadas para el desarrollo de una arquitectura de software son: identificación de los requerimientos arquitectónicos, diseño de la arquitectura, documentación de la arquitectura, evaluación de la arquitectura, y validación de la arquitectura con los diferentes interesados en el sistema que se encuentre en desarrollo.

Considerando lo anterior, podemos observar que el rol del arquitecto de software es crítico y sumamente importante, puesto que requiere de una gran variedad de conocimientos, tales como: ingeniería de requerimientos, teoría de arquitecturas de software, codificación, tecnologías de desarrollo, plataformas de hardware y software.

De igual manera, requiere de saber negociar intereses encontrados de múltiples involucrados en el desarrollo de un sistema de software; promover la colaboración entre el equipo; entender la relación entre atributos de calidad y estructuras; ser capaz de transmitir claramente la arquitectura a los equipos; escuchar, y entender múltiples puntos de vista. El arquitecto de software debe interaccionar con todos los involucrados en el desarrollo de un sistema de software, y ser capaz de dialogar con el analista para obtener los requerimientos significativos, diseñarlos y transmitirlos al programador para su codificación.

Podemos concluir que el diseño de una arquitectura de software debe considerarse una parte fundamental, crítica e imprescindible en el desarrollo de un sistema de software, ya que es precisamente en esta fase en donde recae toda la creatividad, experiencia y creación de la propuesta de solución que más se adecue a las necesidades de nuestro cliente y le permita lograr sus objetivos.

Se trata de un concepto que nació hace ya varios años, no obstante, emerge recientemente como concepto formal, como un proceso de ingeniería. En general, la mayoría no tiene un proceso formal

definido para desarrollar la industria de software y, aunque no es una tarea sencilla el adoptar la creación de una arquitectura de software, se requiere romper paradigmas en la forma de trabajo de las personas. Los profesionales de la industria de software y, específicamente, quienes están dedicados al diseño de sistemas, deben capacitarse ampliamente en el campo de la arquitectura de software para cumplir con esta importante etapa del ciclo de vida de un sistema.

De manera concreta, al diseñar una arquitectura de software se debe crear y representar componentes que interactúen entre ellos y tengan asignadas tareas específicas, además de organizarlos de forma tal que se logren los requerimientos establecidos. Podemos partir con patrones de soluciones ya probados, con la intención de no comenzar de cero las propuestas y utilizar modelos que han funcionado. Estas soluciones probadas se conocen como estilos arquitectónicos, patrones arquitectónicos y patrones de diseño, que van de lo general a lo particular. Un estilo arquitectónico consiste de una colección de tipos de componentes con una descripción del patrón o interacción a través de ellos.

El estilo afecta a toda la arquitectura de software y puede combinarse en la propuesta de solución. Un patrón arquitectónico se enfoca a dar solución a un problema en específico, de un atributo de calidad, y abarca solo parte de la arquitectura. Un patrón de diseño ayuda a diseñar la estructura interna de un componente específico, es decir, su detalle. Aunque estos estilos y patrones se pueden adoptar, también pueden adaptarse con objeto de lograr alguna funcionalidad concreta esperada.

Un aspecto importante en el diseño de la arquitectura es que los atributos de calidad establecidos, determinan los estilos arquitectónicos que pueden ser utilizados o adoptados, en tanto pueden contribuir o afectar el logro de dichos atributos de calidad (fig. 1).

Fig. 1. Estilos

Otro elemento importante dentro de la arquitectura de software es que debe definirse a través de vistas, que representan las diferentes perspectivas de nuestro diseño, como mostrar el diseño de la estructura general de un edificio, junto con el plano de la instalación hidráulica, el plano de la instalación eléctrica, o de la instalación de la red de voz y datos.

Las vistas arquitectónicas pueden representarse mediante lenguajes de modelado, como UML, aunque también existen lenguajes especializados de descripción arquitectónica (ADLs) como ACME, para especificar de manera sintáctica y gráfica los componentes de una arquitectura de software.

La definición de las vistas de la arquitectura de software debe documentarse de manera completa, incluyendo toda la explicación de su diseño, es decir, lo que se ha representado gráficamente, así como las justificaciones de por qué se llegó, fue mejor o se omitieron partes de la propuesta de solución. De la misma manera que existe un documento de especificación de requerimientos de software, se debe crear un documento de la arquitectura de software del sistema deseado, que servirá para generar el diseño detallado de dicho sistema.

Estudio del arte de la arquitectura SOA

Al realizar un análisis de todo lo tratado hasta ahora, más específicamente de los estilos mencionados anteriormente, se ha escogido el estilo arquitectónico orientado a servicios (SOA) como el estilo que regirá la arquitectura de este sistema, esta elección fue basada en el análisis de la naturaleza y las características principales del problema fundamental que enfrenta el proyecto La Casona 23yB.

Por esta razón lleva a cabo un estudio sobre el estado del arte de la Arquitectura Orientada a Servicios (SOA) y los Servicios Web, en el que se presentan todos los componentes que forman la arquitectura así como su organización y su aplicación para resolver el desarrollo de sistemas distribuidos basados en servicios. También se estudian los principales patrones de diseño que se aplican en este tipo de arquitecturas.

A lo largo de los últimos años y sobre todo con la evolución que ha tenido en la informática, las instituciones cada vez necesitan más la posibilidad de integración, y existen multitud de sistemas heterogéneos.

La aparición de los Servicios Web permite tener una arquitectura común para exponer la funcionalidad de las distintas aplicaciones permitiendo la integración de las mismas y de los sistemas, creando nuevos

procesos de negocio que permitan crecer a las plataformas tecnológicas, a los departamentos y por supuesto a las diferentes instituciones.

En cuanto a los Servicios Web es importante tener en cuenta que no son los responsables del comportamiento de un servicio, sino que se centran en como se puede acceder a dicho comportamiento que ofrece un servicio.

Antes de continuar veamos la definición que hace la “World Wide Web Consortium (W3C)” de un Servicio Web:

“Es un sistema software identificado por medio de una URI, que publica interfaces que son definidas y descritas usando XML. Esta definición de interfaz puede ser descubierta por otros sistemas software. Estos sistemas pueden interactuar con el Servicio Web de la manera prescrita en la definición, usando XML, utilizando mensajes cubiertos por los protocolos de Internet.”

Otro concepto que se debe tener claro es el de patrón. Un patrón es simplemente un mecanismo para resolver y tener documentado un problema que ocurre múltiples veces en un determinado contexto. La utilización de patrones permite resolver el problema que dice el patrón a la hora de construir software.

La Arquitectura Orientada a Servicios (SOA) proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

Esta metodología es muy simple, cumple con los estándares más estrictos de programación, se logran aplicaciones robustas, y claro, siempre logrando que los sistemas ya sean para Web o locales sean tan sencillos de usar que no necesiten manual de operación a su vez siendo rápidos y eficientes.

Pasando a características técnicas, estas aplicaciones ya no tienen limite de plataforma (se puede usar Linux, Windows, Macintosh, etc.), se pueden acceder de una manera segura y desde cualquier parte del mundo si sus necesidades así lo requieren, la versatilidad de estas aplicaciones le facilitará a las diferentes instituciones también adaptarse a los nuevos estándares gubernamentales como lo es la facturación electrónica o interconexión con otros sistemas.

La Arquitectura Orientada a Servicios

De acuerdo a analistas de la industria, los conceptos básicos de la Arquitectura Orientada a Servicios (SOA) se establecieron desde hace 20 años. Por tanto, ¿Qué es un servicio en SOA? ¿Cómo puede

ayudar el establecimiento de una Arquitectura Orientada a Servicios?, ¿qué ofrece de nuevo?, ¿por qué esta tecnología tiene tanto éxito mientras otras fallan?

¿Qué es un servicio en SOA?

Un servicio en SOA es una función de aplicación empaquetada como un componente reutilizable para ser usado un proceso de negocio.

El servicio proporciona información o facilita el cambio de datos de negocio de un estado válido y consistente a otro.

Un servicio SOA es autocontenido y sin estado.

La implementación concreta de un servicio SOA no es importante. A través de protocolos de comunicación bien definidos, los servicios pueden ser invocados de manera que se hace hincapié en la interoperabilidad y en la transparencia de localización.

¿Cómo puede ayudar el establecimiento de una arquitectura SOA?

El establecimiento de una arquitectura orientada a servicios puede ayudar a preparar tanto a IT (Tecnologías de Información en español) como a los procesos de negocios para un cambio rápido. Aún en las etapas tempranas de adopción de una SOA, su organización puede beneficiarse.

- Disminuye los tiempos de los ciclos de desarrollo e implementación usando bloques de construcción de servicios reutilizables, prefabricados.
- Permite se integre por toda la empresa (incluso los sistemas históricamente separados) y facilite las fusiones y adquisiciones de empresas.
- Reduce tiempos de ciclo y costos pasando de transacciones manuales a automáticas.
- Facilita la realización de negocios con los asociados de negocios aumentando su flexibilidad.

El enfoque SOA puede colocar un puente entre lo que usted quiere que su negocio cumpla y las herramientas de infraestructura que tiene que tener para llegar allí.

¿Cuáles son los elementos de SOA más importantes para su éxito?

Como primer punto se encuentra la flexibilidad. SOA es la primera arquitectura de Tecnologías de Información (TI) que asume lo que los negocios han sabido desde hace mucho tiempo. Se trata

esencialmente de un set (juego) de servicios sueltos, donde cada uno es relativamente económico para construirlo o reemplazarlo si es necesario. Al ser independientes, el poder unirlos permite a SOA adaptar cambios, cuestión imposible para arquitecturas tradicionales.

En la Arquitectura Orientada a Servicios, se puede reemplazar un servicio sin tener que preocuparse por la tecnología fundamental; la interfase es lo que importa, y está definida en un estándar universal en Servicios Web y XML. Esto es flexibilidad a través de la interoperabilidad. También es la habilidad de asegurar los activos existentes, aplicaciones y bases de datos legales y hacerlos parte de las soluciones empresariales extendiéndolos al SOA en vez de reemplazarlos. El resultado en la red es la habilidad de evolucionar rápida y eficientemente, en otras palabras, adaptarse "orgánicamente" de acuerdo a la demanda del negocio. Esto es realmente nuevo.

En segundo lugar está la relevancia para el negocio. SOA es Tecnologías de Información expresada a un nivel que tiene un significado importante para la colaboración del negocio y profesionales del área. Sus servicios actuales pueden coordinar unidades de trabajo muy cercanas a las actividades del negocio; piense, por ejemplo, en un servicio llamado "Actualización de órdenes de trabajo". Éstos son inmediatamente relevantes para los analistas de la empresa que participan en la creación y definición de nuevos procesos permitiendo el "Servicio Dirigido Empresarial".

Desde que los Servicios Web sustituyen la mayoría de las tecnologías fundamentales, muy poca tecnología de habla es requerida. Los negocios y las TI se enfocan en la lógica del negocio y la comunicación; finalmente comparten el lenguaje de servicios. Esto también es relativamente nuevo y tendrá implicaciones en la entrega de servicios.

¿Cuáles son las principales barreras a vencer para obtener el éxito de SOA?

SOA es un nuevo horizonte para las TI. Como cualquier gran cambio, las principales barreras son organizacionales, no técnicas. A continuación ejemplificaremos algunas:

Administración: Servicios compartidos es lo principal para utilizar SOA. La habilidad para ensamblar rápidamente aplicaciones o procesos está basada en la disponibilidad de algunos servicios que pueden ser compartidos. Hacer esto, por definición, requiere administración.

Desarrollo Cultural: Al utilizar SOA se requiere un cambio significativo en el estilo de programar. Muchos desarrolladores utilizan equipos diferentes para resolver problemas de manera independiente para cada

aplicación. En SOA necesitarán escribir aplicaciones para ser re-utilizadas en mente, usando códigos existentes, a los cuales se podrá tener acceso constantemente.

¿Cómo se distingue actualmente SOA de anteriores estándares de integración y conectividad tales como CORBA?

CORBA era mucho más ambicioso, tecnológicamente hablando, que SOA, y requería una tremenda habilidad de conocimientos en su implementación. Ésta es rara de encontrar, lo cual contribuyó a no entender el significado del estándar. SOA, en contraste, es sencillo y está basado en estándares universales, lo cual asegura que las habilidades para construir aplicaciones sean muy accesibles.

En la Arquitectura Orientada a Servicios, la distribución de los beneficios permite un desarrollo óptimo de funciones como la "Actualización de órdenes de trabajo". Con CORBA, la distribución de los beneficios eran aplicaciones con diferentes propiedades y métodos. Con SOA habrá menor control y poder, pero es más fácil de manejar. Técnicamente no será muy poderoso pero es muy inteligente en el rol de las organizaciones y personas que buscan el éxito en las TI.

Objetivos de una Arquitectura Orientada a Servicio (SOA)

Se deben contemplar entre los objetivos dos diferentes, desde el punto de vista empresarial y desde el punto de vista de la tecnología.

1 Desde el punto de vista empresarial

Cuando una institución decide hacer uso de una arquitectura SOA es porque tiene objetivos específicos de negocio que cubrir, reducir costes, aumentar ingresos, mejorar la productividad, comunicación interempresarial con varias empresas y ajustar los sistemas a los requerimientos del negocio.

También se puede decir que una arquitectura SOA consiste en una forma de modularizar los sistemas y aplicaciones en componentes de negocio que pueden combinarse y recombinarse con interfaces bien definidas para responder a las necesidades de la empresa.

Con el uso de entornos orientados a servicios las diferentes instituciones pretenden mejorar la interacción con los clientes, partners, proveedores, empleados y también reducir el ROI (Return of Investment) retorno de la inversión, es decir, conseguir una mayor rentabilidad de las inversiones tecnológicas.

Para las instituciones se abre un abanico amplio de aplicación, desde la utilización en la cadena de suministro, entornos B2B o servicios de seguridad.

1.1 Beneficios para el negocio

Eficiencia. Transforma los procesos de negocio en servicios compartidos con un menor coste de mantenimiento.
Capacidad de respuesta. Rápida adaptación y despliegue de servicios, clave para responder a las demandas de clientes, partners y empleados.
Adaptabilidad. Facilita la adopción de cambios añadiendo flexibilidad y reduciendo el esfuerzo.

2 Desde el punto de vista tecnológico

Las arquitecturas SOA pretenden concebir las aplicaciones desde otro punto de vista, una aplicación orientada a servicios combina datos en tiempo real con otros sistemas capaces de fusionar los procesos de negocio.
Las aplicaciones basadas en SOA utilizan tecnología totalmente estándar como es XML y Servicios Web para la mensajería. Estándares como SOAP, Web Services Description Language (WSDL) y Business Process Execution Language (BPEL), estandarizan así la compartición de información, el modelo de integración de procesos y la cooperación entre aplicaciones.
Realizando aplicaciones orientadas a servicio se pueden conectar aplicaciones heterogéneas con el aumento de flexibilidad que supone, y un punto muy importante es que permite que las organizaciones interactúen cuando realmente lo requieran, sin necesidad de tener conexiones permanentes.
Como una arquitectura SOA se basa en estándares, el tiempo de aprendizaje de utilización de las tecnologías sobre las que se apoya se reduce drásticamente.

2.2 Beneficios Tecnológicos

- Reduce la complejidad gracias a la compatibilidad basada en estándares frente a la integración punto a punto.
- Reutiliza los servicios compartidos que han sido desplegados previamente.
- Integra aplicaciones heredadas limitando así el coste de mantenimiento e integración.

- Beneficios en el desarrollo, ya que las aplicaciones son reutilizables, más fácil de mantener y tienen la capacidad de ampliación de las funcionalidades del sistema, exponiéndolas de una forma segura.

Fuera de los objetivos desde el punto de vista empresarial y desde el punto de vista de la tecnología, hay que ver que antes de que aparecieran los Servicios Web existían tres técnicas para comunicar aplicaciones:

- Se podía escoger una plataforma particular para ofrecer un servicio, como puede ser la plataforma J2SE con la utilización de RMI (Remote Method Invocation) para implementar el mecanismo de comunicación.
- Se podía utilizar CORBA (Common Object-Request Broker Arquitectura).
- Se podía utilizar un protocolo definido de comunicación particular entre dos aplicaciones.

Por tal motivo se puede ver que el objetivo de una arquitectura SOA es también el de proveer de la transparencia en la localización del Servicio Web, es decir, de la posibilidad de utilizar un determinado servicio que se encuentre en cualquier lugar, sin la necesidad de tener que modificar el código existente.
En el nivel más alto de la arquitectura orientada a servicio se pueden encontrar tres componentes:

- El servicio: Pueden participar cualquier número de servicios. Cada servicio tiene una funcionalidad a la que pueden acceder el resto de servicios y clientes.
- El directorio: El directorio tiene información sobre los servicios y la funcionalidad de estos. Y también tiene la información de cómo se puede acceder a cada servicio.
- El cliente: Usa el directorio para localizar servicios y poder usar su funcionalidad. Un cliente puede ser otro servicio que quiere acceder o utilizar la funcionalidad que aportan otros servicios.

Y también se puede encontrar tres colaboraciones entre los componentes, puede llegar a ser una de las partes más importantes de la arquitectura:

- Localización de servicios: Clientes potenciales de los servicios localizan los servicios por medio del directorio. El directorio aporta a los clientes la información sobre como encontrar un servicio.
- Publicación de servicios: Un componente publica un servicio, haciéndole disponible a los clientes a través del directorio.

- La comunicación entre los servicios y el cliente: El cliente hace peticiones al servicio a través del protocolo de red especificado en la información del servicio que tiene el directorio. El servicio recoge la petición del cliente y le retorna la información pedida.

Lo anterior se puede ver en las siguientes figuras:

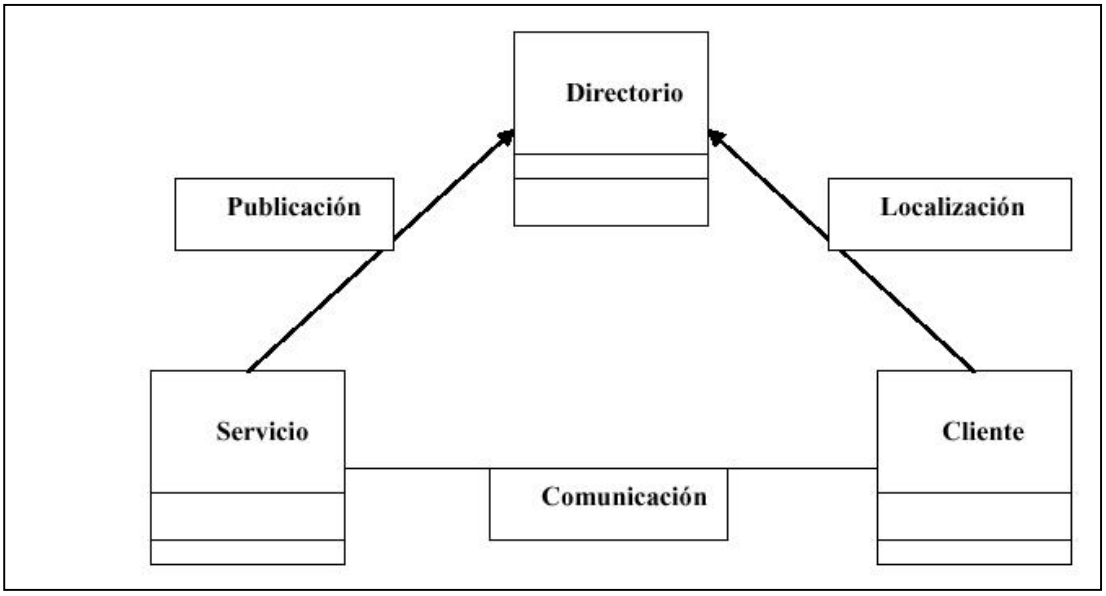

Fig. 2. Componentes de la arquitectura SOA

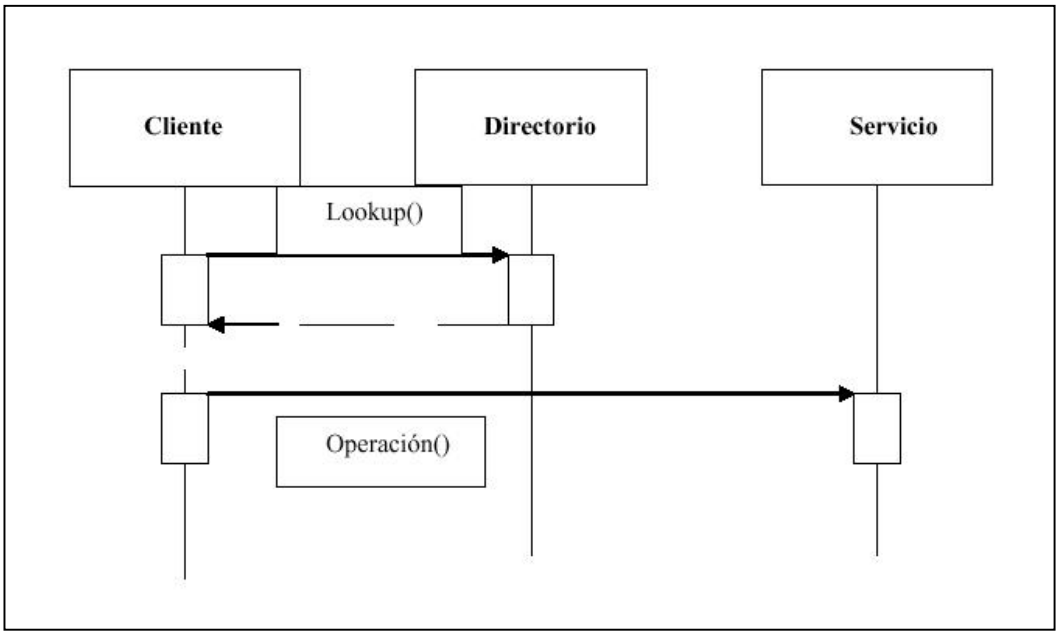

Fig. 3. Diagrama de secuencia de la arquitectura SOA

Normalmente el directorio implementa UDDI (Universal Description, Discovery, and Integration), y todos los componentes se comunican por un protocolo basado en XML (SOAP) y para describir un servicio se utiliza el lenguaje WSDL (Web Service Description Lenguaje).

Patrones SOA

También para la construcción de aplicaciones orientadas a servicio utilizando arquitecturas SOA, se han desarrollado una serie de patrones de software, siguiendo la línea de la banda de los cuatro (GoF).
Estos patrones se pueden dividir en cinco categorías:

Aprendizaje: Es importante entender el entorno de los Servicios Web.
Dentro de esta categoría se puede encontrar:

- Service-Oriented Architecture: Es el patrón que forma la arquitectura de los Servicios Web como ya hemos visto anteriormente.

- Architecture Adapter. Se puede ver como un patrón genérico que facilita la comunicación entre arquitecturas.
- Service Directory: Este patrón facilita la transparencia en la localización de servicios, permitiendo realizar robustas interfaces para encontrar el servicio que realmente se quiere.

Adaptación: Estos patrones son los llamados básicos para conocer el funcionamiento del entorno de los Servicios Web.
En esta categoría se encuentran:

- Business Object: Un business object engloba a un concepto de negocio del mundo real como puede ser un cliente, una compañía o un producto, por ejemplo, y lo que pretende este patrón es trasladar el concepto de objeto de negocio dentro del paradigma de los Servicios Web.
- Business Process: Este patrón se utiliza para tratar con procesos de negocio. En este momento existen dos estándares:
- Business Process Execution Lenguaje (BPEL) propuesto por Bea Systems, IBM y Microsoft.
- Business Process Modeling Languaje (BPML) propuesto por el resto de compañías que no están en el grupo anterior como pueden ser WebMethods, SeeBe-yond, etc.
- Bussines Object Collection: Con este patrón se pueden realizar composiciones de procesos de negocio
- Asynchronous Bussines Process: Este patrón es la evolución del patrón anterior Bussines Process.

Determinando Cambios: Aunque los Servicios Web permiten llamadas asíncronas, las implementaciones del servicio pueden estar basados en paso de mensajes, también son importantes los servicios basados en eventos, estos patrones se basan en patrones tradicionales como el Observer o el patrón Publicación/Suscripción.
En esta categoría podemos encontrar:

- Event Monitor: Es un patrón para crear formas efectivas para integrar aplicaciones sin la intervención de otros componentes. El escenario más común donde se utiliza este patrón es para aplicaciones EAI (Enterprise Application Integration).
- Observer Services: Este patrón representa la manera más natural de detectar cambios y actuar en consecuencia.

- Publish/Subscribe Services: Es la evolución del Observer Pattern, mientras que el patrón Observer se basa en el registro, el patrón Publish/Subscribe se basa en notificaciones, esto permite que distintos servicios puedan enviar la misma notificación.

Redefinición: Estos patrones te permiten acceder al comportamiento de un servicio que está implementado en un lenguaje. Ayudan a entender el entorno del Servicio Web y a moldear este entorno de acuerdo con nuestras necesidades.
En esta categoría podemos encontrar:

- Physical Tires: Este patrón ayuda a estructurar mejor la lógica de negocio de los Servicios Web, e incluso se puede utilizar para controlar el flujo de negociaciones que puede llegar a producirse utilizando el patrón Publish/Subscribe.
- Connector: Este patrón se suele utilizar con el anterior para resolver los posibles problemas que surgen en la subscripción.
- Faux Implementation: Es una alternativa para resolver los problemas que surgen en la utilización de eventos en los Servicios Web. Es simplemente un “socket abierto” que recibe conexiones y aporta las respuestas para los distintos eventos.

Creando flexibilidad: Para crear servicios más flexibles y optimizados.

En esta categoría se encuentran:

- Service Factory: Es uno de los patrones más importantes y permite la selección de servicios y aporta flexibilidad en la instanciación de los componentes que crean los Servicios Web. Este patrón también se suele utilizar con el patrón Service Cache para aportar una mayor flexibilidad en el mantenimiento de las aplicaciones que utilizan Servicios Web, aportando un mayor ROI a las aplicaciones.
- Data Transfer Object: Este patrón aporta rendimiento ya que permite recoger múltiples datos y enviarlos en una única llamada, reduciendo el número de conexiones que el cliente tiene que hacer al servidor.

- Partial Population: Este patrón permite a los clientes seleccionar únicamente los datos que son necesarios para sus necesidades y sólo recuperar del servidor lo necesario. Este patrón además de rendimiento porta mayor ancho de banda en la red.

Algunos patrones utilizan otros por ejemplo el Business Process Patern usa el Business Object Patern y el Business Object Collection. Y el Service-Oriented Architecture usa los patrones Service Directory y Architecture Adapter.

Servicios Web y arquitecturas orientadas a los servicios

Las herramientas disponibles para el diseño de la arquitectura han sufrido un gran resquebrajamiento con el surgimiento de los Servicios Web y de la arquitectura manejada por servicios. Los Servicios Web constituyen una implementación de los componentes en el mundo Web.
La definición de Servicio Web que el Grupo de Arquitectura de Servicio Web del W3C da es la siguiente:
Un Servicio Web es una aplicación de Software identificada por una URI, cuya interfaces y ligadura (una conexión lógica entre un proceso en el cliente y un proceso en el servidor) son capaces de ser definidas , descritas y descubiertas por artefactos XML y soporta interacciones directas con otras aplicaciones de software usando mensajes XML vía protocolos basados en Internet.
Un Servicio Web no requiere un mecanismo de descripción de alguna clase tal como WSDL, no requiere SOAP ni http. Si el Servicio Web no esta basado en XML se hace más complejo y menos abierto.
Los beneficios de los Servicios Web se mencionan a continuación:

- Ayudan a reducir la complejidad ya que encapsulan el proceso del negocio en componentes reutilizables.
- Ayudan a mejorar la interoperatividad al actuar como intermediario entre las aplicaciones heredadas o realizadas sobre plataformas específicas.
- Permiten componer con otras partes funciones del negocio de alto nivel.
- Ayudan a ocultar detalles de la implementación del back-end a través del uso de definiciones de la interfaz bien conocidas y estandarizadas.
- Permiten la integración justo en tiempo al promover la perdida de acoplamiento y el enlace tardío

- Constituyen plataformas de implementación neutrales lo que promueve una verdadera interoperatividad. Por lo que usar Servicios Web XML permite integrar con otras plataformas y tecnologías.
- Cumplen los estándares de Internet establecidos.
- La seguridad puede implementarse de forma más simple.
- El Servicio Web no posee interfaz usuario sino solo interfaz bajo control de programa o programática. Lo anterior facilita la reutilización de estas componentes puesto que se pueden enlazar con otros componentes y vincularlos a disímiles interfaces usuario. Por tanto, estas componentes permiten una utilización muy flexible. Esto ha habilitado también una forma de vender y reutilizar componentes en Internet.
- La comunicación a través de mensajes disminuye el acoplamiento, aumenta la disponibilidad y escalabilidad.

Sin embargo, el uso de los Servicios Web aumenta la lentitud de la aplicación resultante por lo que debe ser analizado en el caso de aplicaciones simples su conveniencia de uso.

Se afirma por muchos que esta es la próxima etapa en la evolución de compartir la información. No es un secreto, que las grandes empresas manufactureras pueden integrar sus sistemas internos a los de sus socios, para automatizar su proceso del negocio de manera de ahorrar dinero, reducir los ciclos, mejorar la productividad y ser exitosos. Pero la integración ha sido tradicionalmente cara, ha consumido mucho tiempo y solo alcanza tal profundidad su utilización en la cadena de suministro o en los canales con los socios. La industria necesita una forma fácil, menos costosa y efectiva de establecer sus conexiones. Los Servicios Web llenan esta visión de conexiones fáciles, menos costosas y más efectivas.

Una arquitectura orientada a servicios (SOA: Service Oriented Architecture) es un método de diseñar y construir soluciones de software poco acopladas que expongan sus funciones del negocio como servicios de software accesibles programáticamente para ser usadas por otras aplicaciones a través de la publicación de interfaces descubribles. Los Servicios Web representan una implementación de una arquitectura orientada a servicios pero no todas las aplicaciones SOA pueden ser consideradas Servicios Web.

El primer reto de los Servicios Web ha sido muy bien cumplido con la inclusión de estándares como el lenguaje XML (extensible markup language), el lenguaje de descripción de los Servicios Web WSDL (Web

services description language) y el protocolo de acceso a los objetos SOAP (Simple Object Access Protocol)), junto a algunos especificaciones opcionales similares a la descripción universal, descubrimiento e integración (UDDI). Estas tecnologías permiten que las aplicaciones publiquen datos como Servicios Web a ser consumidos por otras aplicaciones.

Uno de los próximos pasos para construir más sofisticados y valiosos Servicios Web es crear estándares para asegurar interoperatividad y orquestación en la interacción entre el Servicio Web y el proceso del negocio informatizado.

Los Servicios Web afectarán la implementación del negocio, la integración y la aplicación de patrones donde ellos están en la frontera (entre negocios, entre aplicaciones, entre componentes lógicas de la solución, etc.) a lo largo de la cual la información debe ser intercambiada.

El surgimiento de los Servicios Web ha provocado la definición de un estilo de arquitectura que no estaba contemplado a la escala debida en los ADLs. Se requiere satisfacer requisitos descriptivos de alto nivel de abstracción que las herramientas basadas en objeto en general y UML en particular no cumplen satisfactoriamente. Los ADL poseen tuberías (pipes: Conexión de programa temporal entre dos programa o comandos) y filtros (procesador que soporta restricciones sobre particulares comandos y datos), repositorio (BD), eventos, capas, llamada y retorno/OOP y máquinas virtuales.

Pero la descripción de sistemas basados en componentes presenta también limitaciones serias, que no son de detalle sino más bien estructurales entre las que se encuentran:

- Sólo proporcionan una única forma de interconexión primitiva: la invocación de método. Esto hace difícil modelar formas de interacción más ricas o diferentes
- El soporte de los modelos OO para las descripciones jerárquicas es, en el mejor de los casos, débil.
- No se soporta la definición de familias de sistemas o estilos
- No hay recurso sintáctico alguno para caracterizar clases de sistemas en términos de las restricciones de diseño que debería observar cada miembro de la familia.
- No brindan soporte formal para caracterizar y analizar propiedades no funcionales, lo que hace difícil razonar sobre propiedades críticas del sistema, tales como desempeño y robustez.

Jørgen Thelin, alega que el período de gloria de la programación orientada a objetos (OOP) podría estar acercándose a su fin. El lenguaje UML posee limitaciones:

- No brinda medios para detectar tempranamente errores de requisito y diseño.
- Aunque es posible representar virtualmente cualquier cosa, incluso fenómenos y procesos que no son software, muchas veces no existen formas estándares de materializar esas representaciones, de modo que no pueden intercambiarse modelos entre diversas herramientas y contextos sin pérdida de información.
- Ni las clases, ni los componentes, ni los paquetes, ni los subsistemas de UML son unidades arquitectónicas adecuadas: las clases están tecnológicamente sesgadas hacia la Orientación a Objetos y representan entidades de granularidad demasiado pequeña; los componentes "representan piezas físicas de implementación de un sistema" y por ser unidades de implementación y no de diseño, es evidente que existen en el nivel indebido de abstracción; un paquete carece de la estructura interna necesaria; los subsistemas no pueden aparecer en los diagramas de despliegue, ni se pueden mezclar con otras entidades de diferentes granos y propósito, carecen del concepto de puerto y su semántica y pragmática se han estimado caóticas.
- La falta de modelos causales rigurosos; aunque UML proporciona herramientas para modelar requerimientos de comportamiento (diagramas de estado, de actividad y de secuencia), al menos en UML 1.x no existe diferencia alguna, por ejemplo, entre mensajes opcionales y mensajes requeridos; se pueden "anotar" los mensajes con restricciones, por cierto, pero es imposible hacerlo de una manera estándar.
- Aunque los objetos de UML se pueden descomponer en piezas más pequeñas, no es posible hacer lo propio con los mensajes.
- Los modelos de implementación son inmaduros y no hay una clara diferenciación o un orden claro de correspondencias entre modelos notacionales y meta-modelos, o entre análisis y diseño.
- En el meta-modelo de UML se definen numerosos conceptos que no aparecen en los modelos de implementación, que usualmente son lenguajes orientados a objeto; "Agregación", "Asociación", "Composición" y "Restricción", "Estereotipo" son ejemplos de ello.
- Herramientas de CASE y modelado industriales de alta calidad, como Rational Rose, TogetherSoft Together o JBuilder de Borland o productos de código abierto como ArgoUML, no poseen definiciones claras de las relaciones de clase binarias como asociación, agregación o composición; distinguen entre ellas a nivel gráfico, pero el código que se sintetiza para las diferentes clases binarias es el mismo. Las herramientas de reingeniería producen por lo tanto relaciones erróneas e

inconsistentes; simplemente generando código a partir del diagrama y volviendo a generar el diagrama a partir del código se obtiene un modelo distinto.

UML no es un ADL en el sentido usual de la expresión. Los manuales contemporáneos de UML carecen del concepto de estilo y sus definiciones de “arquitectura” no guardan relación con lo que ella significa en el campo de los ADLs.

Tecnologías

Selección de la metodología a utilizar:

Luego de un profundo análisis sobre las diferentes metodologías de desarrollo de software, llegamos a la conclusión de escoger RUP, pues esta metodología recoge lo mejor de cada una de las analizadas y traza una mejor y completa línea de trabajo, es un proceso de desarrollo de Software que proporciona una guía en el orden de las actividades de un equipo, dirige las tareas individuales de los desarrolladores, especifica que productos deberían ser desarrollados y ofrece criterios para monitorear y medir los productos y actividades de un proyecto así como usar casos de uso en forma efectiva facilita tener una interacción continua y clara con el cliente, evitando construir sistemas de información que no están acorde a las expectativas finales. Ver Anexo #1.

Selección del Lenguaje de programación a utilizar:

Se utilizara como lenguaje de programación PHP, el cual se ejecuta del lado del servidor, pues reúne las mejores capacidades funcionales para la creación de la aplicación Web, así como utilizar la capa de abstracción de datos ADOdb para acceder a la base de datos, con posibilidad de migrar a diferentes gestores de bases de datos por si en un futuro el cliente desea cambiar su gestor, también existe mucha información, documentación y ejemplos disponibles en Internet, es de la filosofía de Software Libre, pues no tenemos que pagar por licencias o patentes así como actualizaciones.

Como lenguaje del lado del cliente se propone el uso de Java Script, que es un lenguaje orientado a eventos e interpretado, el cual no requiere complicación, este lo utilizaremos para validar todos los datos que sean entrados por el cliente no sean incorrectos, para de esta forma no sobrecargar el servidor y tener que evitar mayor procesamiento de instrucciones. Ver Anexo #1.

Selección del CMS a utilizar:

Para este proyecto se opta por trabajar con el CMS Drupal implementado en PHP al igual que los servicios que se consumen. Este ha sido diseñado desde el principio para ser multi-plataforma. Puede funcionar con Apache o Microsoft IIS como servidor Web y en sistemas como Linux, BSD, Solaris, Windows y Mac OS X. Drupal incorpora una capa de abstracción de base de datos que actualmente está implementada y mantenida para MySQL y PostgresSQL, aunque permite incorporar fácilmente soporte para otras bases de datos, esto es muy importante por si en un futuro se desea migrar hacia otro que no sea el que se este utilizando. Ver Anexo #1.

Selección del Sistema de Gestor de Base de Datos:

Con lo anterior analizado, optamos por trabajar con el SGBD MySQL, pues por sus características de rápido acceso a los datos, es la combinación perfecta con PHP, y existe mucha información y documentación en Internet sobre este gestor. Ver Anexo #1.

CAPÍTULO 2: REPRESENTACIÓN DE LA ARQUITECTURA

Representación de la Arquitectura

Este documento presenta la arquitectura mediante una serie de vistas: Casos de Uso, Lógica, Procesos, Implementación y Despliegue. Para representar estas vistas se ha utilizado el Lenguaje de Modelado Unificado (UML) y se han desarrollado usando Visual Paradigm, de forma que estas permitan visualizar, entender y razonar sobre los elementos significativos de la arquitectura e identificar las áreas de riesgo que requieren mayor detalle de elaboración. Este documento es una forma de comunicar el modelo del sistema, presentando la información y discusiones estructuralmente.
La arquitectura del sistema se descompone en las siguientes dimensiones:

Requerimientos: Requerimientos funcionales y no funcionales del sistema.
Elaboración: Representación lógica del sistema y representación del tiempo de ejecución.
Implementación: Vista de módulos implementados, potenciales escenarios de infraestructura y el deployment (despliegue) de los módulos.

La siguiente sección detalla las vistas de la arquitectura que serán utilizadas para cubrir las dimensiones mencionadas. La sección posterior presenta el framework arquitectónico utilizado.

Representación

La arquitectura del Sistema esta representada siguiendo las recomendaciones de la ayuda extendida del Rational Unified Process. Las vistas necesarias para especificar dicho sistema se enumeran a continuación:

Vista de Casos de Uso: Describe el proceso de negocio más significativo y el modelo del dominio. Presenta los actores y los casos de uso para el sistema.
Vista Lógica: Describe la arquitectura del sistema presentando varios niveles de refinamiento. Indica los módulos lógicos principales, sus responsabilidades y dependencias.
Vista de Procesos: Describe los procesos concurrentes del sistema.

Vista de Implementación: Describe los componentes de despliegue construidos y sus dependencias.

Vista de de Despliegue: Presenta aspectos físicos como topología, infraestructura informática, e instalación de ejecutables. Incluye además plataformas y software de base.

Framework Arquitectónico

La arquitectura sigue el framework "4+1"; este framework define 4 vistas para la arquitectura en conjunto con los escenarios, y es presentado en la siguiente figura.

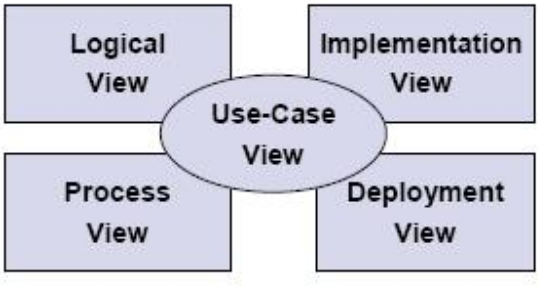

Fig. 5. Las 4+1 Vistas de RUP

Framework 4+1	**Arquitectura**
Vista de Casos de Uso	Casos de Uso, Restricciones y QoS
Vista Lógica	Lógica
Vista de Procesos	Procesos
Vista de Implementación	Implementación, Datos
Vista de Despliegue	Deployment

Tabla2: Mapeo de las vistas utilizadas a las propuestas en el framework

Metas Arquitectónicas y Restricciones

Existen determinados requisitos no funcionales que tienen un impacto significativo en la arquitectura, ellos son:

Software:

- Los Servicios Web están desarrollados sobre el lenguaje de programación PHP 5.0 y PHP 4.0.
- Los Servicios Web utilizarán como base de datos MySQL.

- Se utilizara como servidor Web Apache 2.0.
- La intranet será montada sobre el CMS Drupal.

Usabilidad:

- El uso de Servicios Web nos permite que la aplicación pueda consumir dichos servicios aunque esta este escrita en una plataforma de desarrollo diferente a estos, pues su protocolo de comunicación esta basado en XML y es compatible para todas las plataformas.
- El uso de los servicios Web es muy práctico ya que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro.

Confiabilidad:

- La aplicación debe estar disponible las 24 horas de forma tal que se pueda acceder a todos sus Servicios, y así aprovechar todas sus funcionalidades.

Importación y exportación de datos:

- El sistema deberá almacenar todos los datos en una base de datos MySQL Server, donde puedan ser accedidos por los Servicios.

Portabilidad:

- La portabilidad se observa en los Servicios Web, los cuales podrán ser accedidos desde cualquier protocolo de trasporte que sea capaz de transportar texto, como por ejemplo (http, https, SMTP, etc.). El sistema utilizara http sobre TCP en el puerto 80. Esto se hace debido a que en esta como en toda organización se protege las redes mediante firewalls y se cierran casi todos los puertos TCP salvo el 80 que nunca es bloqueado porque es el que utilizan los navegadores.
- El diseño del Servicio Web permite que el mismo pueda funcionar tanto en sistemas operativos libres como propietarios.
- El sistema puede migrar a varios Sistemas Gestores de Base de Datos debido a que utiliza ADOdb.

Seguridad y Privacidad:

- El acceso será controlado con nombres de usuario y contraseñas. Solo los usuarios con derechos de administrador podrán acceder las funciones administrativas, los usuarios normales no podrán.
- Se establecerá una llave privada entre el cliente y el proveedor del Servicio Web, para evitar acceso no autorizado a los servicios.

Restricciones en el diseño y la implementación:

- Se usa el lenguaje de programación PHP tanto para los servicios como para la aplicación que los consume.
- Se usa la librería NuSoap para la creación de los servicios Web.
- Se usa la librería ADOdb para la abstracción a datos.

Legales:

- La aplicación con todos sus módulos y toda la documentación generada pertenecen al proyecto del Centro Rector de universidad para todos y a la Universidad de las Ciencias Informáticas.

Hardware:

- El servidor donde estarán montados los servicios requiere como mínimo de RAM 1 Gb.
- El servidor de base de datos requiere como mínimo 5 GB para almacenar la Base de Datos.

Vistas Arquitectónicas: Introducción

A partir de aquí se trataran un grupo de diferentes vistas de la arquitectura, las cuales facilitan la comprensión de la misma desde diferentes puntos de vista. Estas son significativas para nuestro proyecto y fueron elegidas de un grupo de vistas candidatas, teniendo en cuenta cuanto detalle necesita cada interesado de cada interés, ya que estas están dirigidas a usuarios específicos para poder cumplir con los objetivos trazados en el trabajo. Es importante tener en cuenta que la arquitectura de un sistema consta de múltiples vistas, asociadas a diferentes dimensiones o estructuras del sistema, además que las vistas se encuentran dirigidas a usuarios particulares y asociadas a requisitos no funcionales concretos, así como que ninguna vista particular constituye la arquitectura del sistema.

Vista de Casos de Uso

Vista del Modelo de casos de uso

En la vista del Modelo de casos de uso se representan los actores y los casos de uso arquitectónicamente significativos del sistema. Estos casos de uso presentan una prioridad crítica para el usuario o encierran una funcionalidad central del sistema.

Actores

Usuario: Existen tres tipos de usuario, usuario común los cuales son generalmente los profesores de los Canales Educativos y Universidad para Todos que acceden al sistema para obtener información y preparar sus clases, además para solicitar los distintos servicios que brinda la institución. Además se encuentran los trabajadores de la institución los cuales son usuarios con funciones específicas dentro del sistema y que tienen diferentes permisos y pueden obtener información del mismo. También se encuentra el administrador que tiene el control total del sistema.

Administrador: Es el que controla todo el sistema en general, es una especialización de todos los usuarios además de tener los privilegios únicos que tiene un administrado en un sistema como son por ejemplo: eliminar, crear, modificar etc.

Funcionaria: Es la encargada de insertar nuevos usuarios al sistema

Diagrama de los casos de uso arquitectónicamente significativos

Fig. 6. Diagrama de los casos de uso arquitectónicamente significativos

Casos de Uso Relevantes de la arquitectura

Solicitar Impresión y Fotocopia

Descripción: El usuario solicita la impresión o fotocopia de determinado documento mediante la intranet de la institución, recibiendo posteriormente de parte del encargado de dicha tarea la confirmación o no de su pedido mediante un correo electrónico, en caso de confirmación se le informará a través de este mismo medio, la forma en que se va a realizar su solicitud.

Pre-condiciones: El usuario debe estar autenticado para poder solicitar cualquiera de estos servicios.

Solicitar Digitalización de Documentos

Descripción: El usuario solicita la digitalización de determinado documento mediante la intranet de la institución, recibiendo posteriormente de parte del encargado de dicha tarea la confirmación o no de su pedido mediante un correo electrónico, en caso de confirmación se le informará a través de este mismo medio, la forma en que se va a realizar su solicitud.

Pre-condiciones: El usuario debe estar autenticado para poder solicitar cualquiera de estos servicios.

Solicitar Transferencia de Formato

Descripción: El usuario solicita la transferencia de formatos como es la solicitud de quemado de CD mediante la intranet de la institución, para así poder extraer información demasiado grande. El usuario recibe posteriormente de parte del encargado de dicha tarea la confirmación o no de su pedido mediante un correo electrónico, en caso de confirmación se le informará a través de este mismo medio, la forma en que se va a realizar su solicitud.

Pre-condiciones: El usuario debe estar autenticado para poder solicitar este servicio.

Solicitar Tiempo de Máquina

Descripción: El usuario solicita el tiempo de maquina mediante la intranet de la institución o mediante un correo electrónico, el sistema le debe mostrar una aplicación donde se registra los puestos de trabajo en

uso y/o reservados, el sistema una vez hecha la reservación le muestra un mensaje informándole que ya que ha sido efectuada la solicitud del tiempo de máquina exitosamente.

Pre-condiciones: El usuario debe estar autenticado para poder solicitar este servicio.

Solicitar Local de Trabajo

Descripción: El usuario solicita un local de trabajo en la institución mediante la intranet, recibiendo posteriormente de parte del coordinador de la institución la confirmación o no de su pedido mediante un correo electrónico, en caso de confirmación se le informará a través de este mismo medio toda información necesaria que debe conocer el usuario.

Pre-condiciones: El usuario debe estar autenticado para poder solicitar este servicio.

Solicitar Préstamo de PC Portátil

Descripción: El usuario solicita el préstamo de PC portátil a la institución mediante la intranet, luego se muestra un mensaje informándole que ya a sido efectuada la solicitud del préstamo, recibiendo posteriormente de parte del coordinador de la institución la confirmación o no de su pedido mediante un correo electrónico, en caso de confirmación se le informará a través de este mismo medio toda información necesaria que debe conocer el usuario para obtener este préstamo.

Pre-condiciones: El usuario debe estar autenticado para poder solicitar este servicio.

Solicitar Acceso

Descripción: El usuario llega a la puerta y solicita el acceso a la institución, específicamente a un área determinada, informándole al agente de seguridad que se encuentre en la puerta que va a recibir servicios, este verificará si tiene acceso y le informará al usuario. En caso de tener acceso, registrará al usuario y le devolverá el solapín que lo identificará y le dará el permiso a estar en la institución. En caso de que no haya estado previamente registrado, el agente de seguridad contacta vía teléfono al Departamento de Gestión y Servicios de Información donde recibe confirmación, si le permiten el acceso, este registrará los datos personales del usuario y le permite el acceso entregándole un solapín. En caso de ser trabajador el portero verificara sus documentos y lo registra en el sistema, permitiéndole la entrada.

Pre-condiciones: Si es usuario, este debe estar previamente en la lista de visitantes a la institución para poder solicitar entrar a la misma, pero en caso de ser trabajador debe traer su identificación.

Gestionar niveles de acceso

Descripción: El administrador es el encargado de realizar operaciones específicas en el sistema, como son, asignar nivel de acceso por área al personal (trabajadores y profesores) permanentes de la Casona, modificar el nivel acceso, o eliminar el nivel de acceso. Una vez realizada cualquiera de estas operaciones finaliza el caso de uso.

Pre-condiciones: El administrador debe estar previamente autenticado como tal para poder tener los privilegios de asignar niveles de accesos por áreas a personal permanente de la Casona (trabajadores y profesores) y modificar o eliminar dichos niveles de acceso.

Consultar Catálogo en Línea (Catalogo OPAC)

Descripción: El usuario accede al catálogo en línea mediante la intranet de la institución, esto le permite al usuario consultar con mayor eficacia y rapidez los registros de los documentos que se encuentran en el fondo de información, el sistema le muestra una interfaz con los materiales que se encuentran para ser consultados.

Pre-condiciones: El usuario debe estar autenticado para poder acceder a este servicio.

Consultar Recurso

Descripción: El usuario accede a la intranet y llena la solicitud donde se registra el préstamo en sala, este servicio se brinda con el fin de posibilitar el acceso directo y consulta de documentos, de materiales disponibles en el fondo de la institución que será de estantería abierta, esto le permite al usuario localizar el documento que necesita sin intermediario. Los materiales pueden estar en la institución o fuera de ella.

Pre-condiciones: El usuario debe estar autenticado para poder acceder a este servicio.

Gestionar usuario

Descripción: La funcionaria accede a la aplicación donde tiene las opciones de registrar usuario, eliminar usuario o actualizar datos del usuario. Una vez que realiza uno de estos servicios el sistema retorna un mensaje indicando al usuario si se realizo o no la operación, finalizando el caso de uso.

Pre-condiciones: La funcionaria debe estar autenticada como tal para poder tener los privilegios de insertar, eliminar o modificar datos del usuario.

Vista lógica

Descripción de la arquitectura vertical de alto nivel

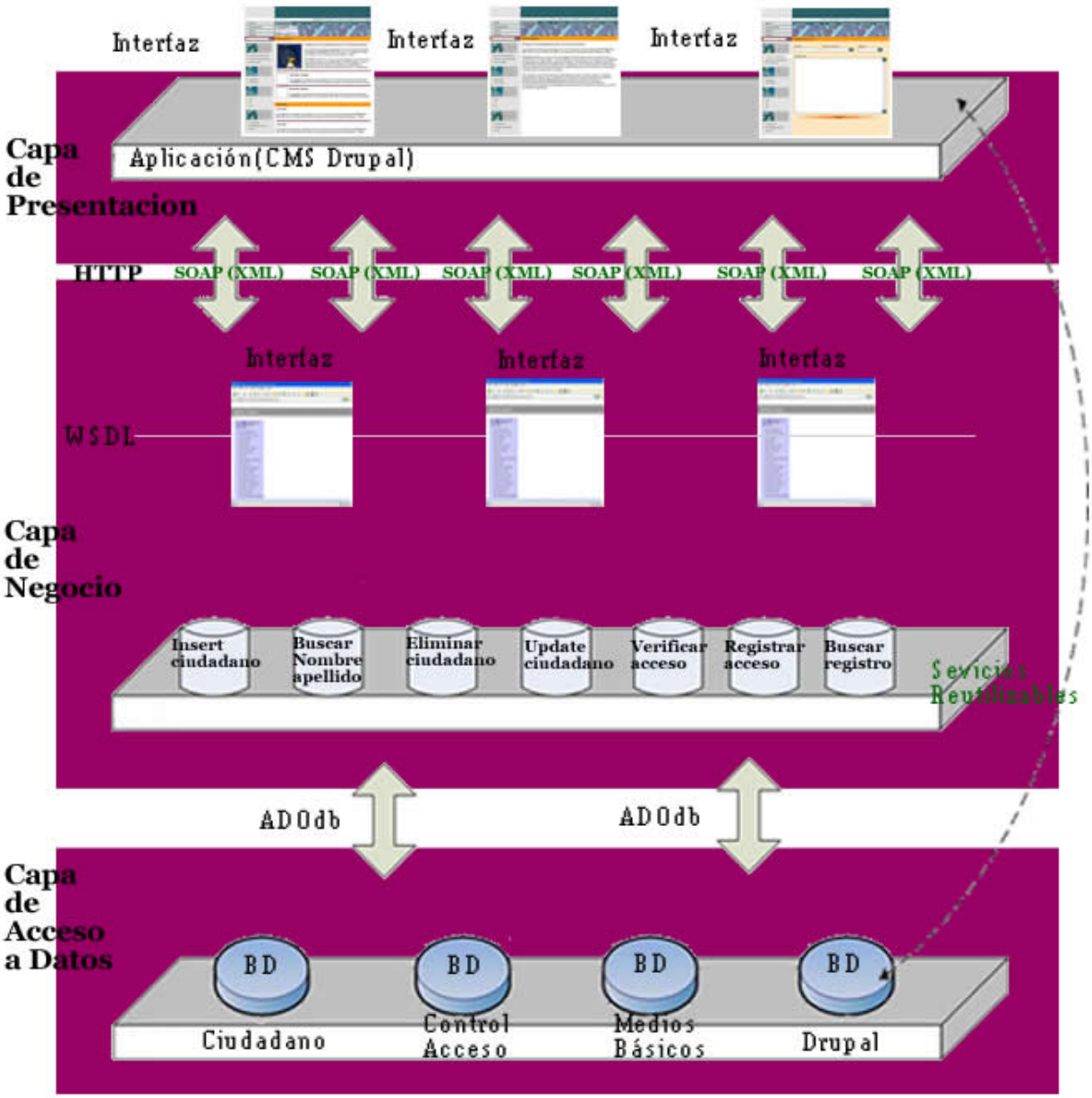

Fig.7. Descripción de la arquitectura vertical de alto nivel

Breve descripción de las capas

Capa de Presentación

En la Capa de Presentación es donde está la aplicación Web montada en el CMS Drupal, esta capa maneja la interacción entre el usuario y la aplicación, además recoge la entrada de los usuarios y controla la navegación por las páginas de la aplicación, no permite la entrada del usuario a la capa de negocio ya que debido a que como lo que se desea implementar son Servicios Web, estos describen su interfaz en WSDL y exponen su funcionalidad recibiendo/enviando mensajes SOAP (que tienen como base XML) a través del protocolo de transporte HTTP.

Capa de Negocio

En la Capa de Negocio es donde están todos los Servicios Web implementados, esta capa recibe las peticiones de la Capa de Presentación mediante mensajes SOAP, en estos mensajes está la dirección URL de los servicios a los cuales se quiere acceder, esta capa es la encargada de interpretar estos mensajes y así ofrecerle los servicios que el usuario desea, estos servicios describen su interfaz en WSDL.

Capa de Acceso a Datos

La Capa de Acceso a datos contiene los servicios de acceso a datos de la aplicación, recibe peticiones de la Capa de Negocio y es la encargada de interactuar directamente con la base de datos.

Descripción de la arquitectura horizontal

Los casos de uso del proyecto se agruparon en cinco módulos atendiendo a sus funcionalidades.

Módulo Acceso:

En este módulo se encuentra todo lo referente al acceso de los usuarios a la institución.

Casos de Uso Contenidos:

Solicitar Acceso.

Gestionar niveles de acceso.

Módulo Gestión de Usuarios:

En este módulo se encuentra todo lo referente a la gestión de usuarios o sea eliminar un usuario, insertar un nuevo usuario, modificar datos de un usuario ya existente y buscar un usuario atendiendo a diferentes patrones de búsqueda.

Casos de Uso Contenidos:

Gestionar Usuario

Módulo Consulta de Recursos:

En este módulo se encuentra todo lo referente a la consulta de los recursos, o sea, recursos para su utilización en la preparación de las clases u otras actividades investigativas por parte de los profesores y otros usuarios que accedan al centro, estos recursos pueden estar en la red o no.

Casos de Uso Contenidos:

Consultar Recursos
Consultar Catálogo Online

Módulo Reproducción de Documentos*:*

En este módulo se encuentra lo referente a la reproducción de diferentes tipos de documentos, ya sea mediante impresión y fotocopiado, quemado de discos o DVD, transferencia de un formato a otro, etc.

Casos de Usos Contenidos:

Solicitar Impresión y Fotocopia
Solicitar Digitalización de Documentos
Solicitar Transferencia de formato

Módulo Reservación:

En este módulo se encuentra lo referente a las reservaciones que realizan los profesores en la institución, ya sea reservación de un local de trabajo, de una PC, etc.

Casos de Uso Contenidos:

Solicitar Tiempo de Máquina
Solicitar Local de Trabajo
Solicitar Préstamo de PC Portátil

Vista refinada de la arquitectura del sistema

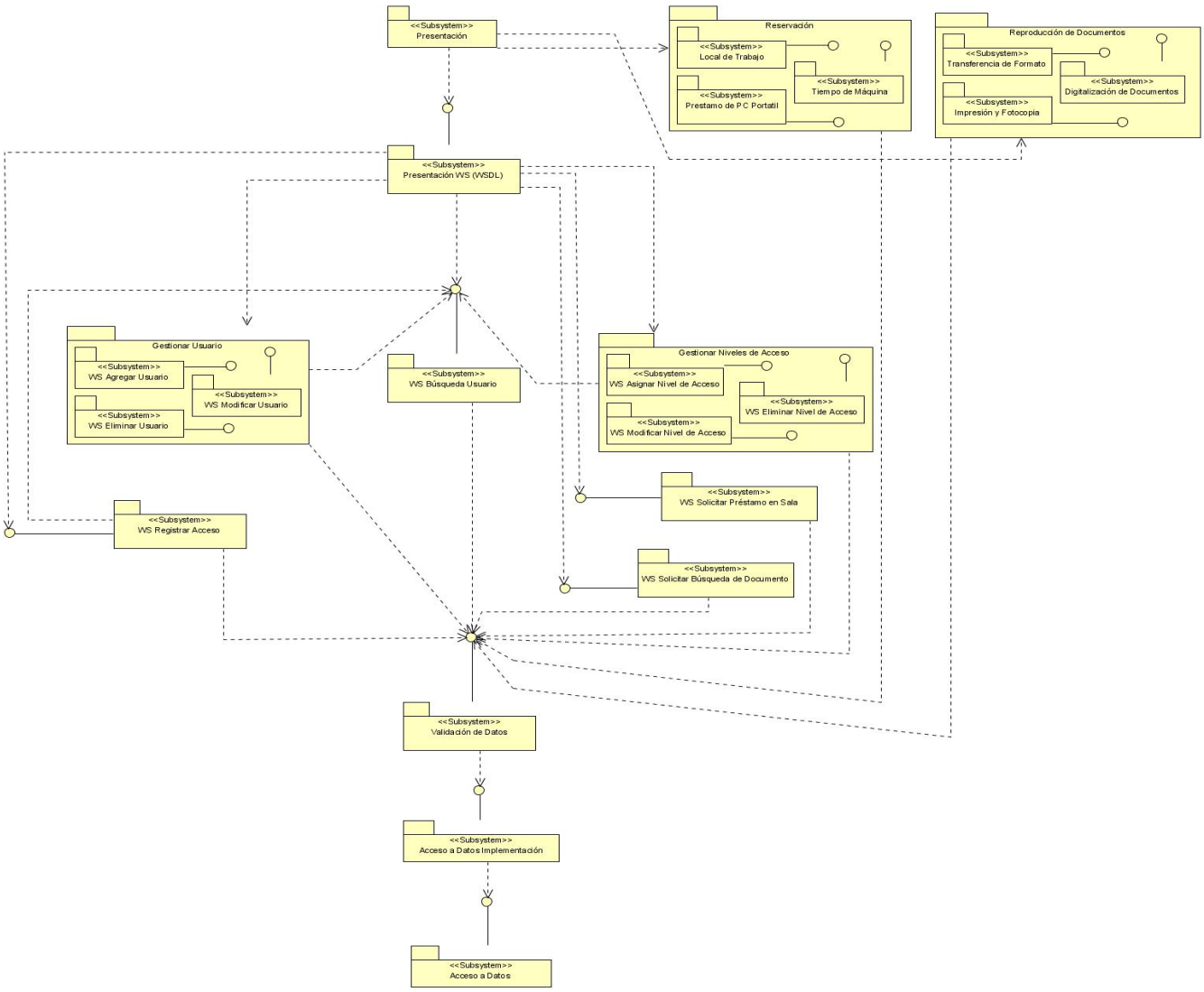

Fig.8. Descomposición en subsistemas

Vista Lógica:

La Vista Lógica es una representación de los Principales subsistemas y paquetes y las relaciones que se establecen entre estos. Estos paquetes encapsulan subsistemas de acuerdo a sus funcionalidades para facilitar la comprensión de la estructura, y a su vez los subsistemas encapsulan clases que determinan

estas funcionalidades como por ejemplo cada uno de los Servicios Web que brindan su funcionalidad para atender una petición del cliente.

Subsistema de Presentación:

Este subsistema interactúa directamente con el usuario, es el encargado de mostrarle una serie de páginas WEB dinámicas donde se encuentran los formularios para introducir los datos necesarios y acceder a los servicios que se brindan. Estas páginas son hechas en un Sistema Manejador de Contenido (CMS), en específico Drupal. Por lo general son variables, en dependencia de los servicios que solicita el usuario.

Subsistema de Presentación del Servicio Web (WSDL):

En este subsistema se encuentran los Servicios Web que se han desarrollado, y el Lenguaje de Descripción de los Servicios Web (WSDL), dicho WSDL representa la interfaz a través de la que va a interactuar el Servicio Web. La comunicación con el Subsistema de Presentación se realiza a través del protocolo SOAP que envía mensajes SOAP con los datos necesarios para el Servicio Web. Este subsistema es el encargado de enviar al usuario las respuestas de los servicios a través de mensajes XML.

Subsistema de Búsqueda de Usuario (WS)

Este subsistema encierra un Servicio Web, que brinda la posibilidad de buscar usuarios en la base de datos de la institución. Es uno de los subsistemas más importantes debido a que otros subsistemas utilizan su funcionalidad.

Paquete Gestionar Usuario

Este paquete está conformado por tres subsistemas que ofrecen funcionalidades relacionadas con la gestión de usuario, Subsistema Eliminar Usuario, Subsistema Agregar Usuario y Subsistema Modificar Usuario. Tiene relaciones con los subsistemas Búsqueda de Usuario y Presentación WS (WSLD) que están dadas por las relaciones entre estos dos subsistemas y los subsistemas que encapsula.

Subsistema Eliminar Usuario (WS)

Encapsula un Servicio Web que se encarga de eliminar usuarios que están registrados como miembros de la institución. Este subsistema depende del subsistema Búsqueda de Usuario.

Subsistema Agregar Usuario (WS)

Subsistema que define la funcionalidad del Servicio Web utilizado para crear un nuevo usuario a la institución y guardar sus datos en la base de datos. Este subsistema utiliza funcionalidades brindadas por el subsistema Búsqueda de Usuario.

Subsistema Modificar Usuario (WS)

Subsistema asume las responsabilidades del Servicio Web utilizado para modificar los datos de un usuario determinado que se encuentre registrado en la base de datos de la institución. Este subsistema depende del subsistema Búsqueda de Usuario.

Paquete Gestionar Niveles de Acceso

Este paquete esta conformado por tres subsistemas que son a la vez tres Servicios Web y los mismos son, Subsistema Asignar Nivel de Acceso, Subsistema Eliminar Nivel de Acceso, Subsistema Modificar Nivel de Acceso, estos subsistemas tienen en común que son utilizados para controlar los niveles de acceso de un usuario determinado de la institución.

Subsistema Asignar Nivel de Acceso (WS)

El paquete Gestionar Niveles de Acceso está conformado por tres subsistemas, Subsistema Asignar Nivel de Acceso, Subsistema Eliminar Nivel de Acceso, Subsistema Modificar Nivel de Acceso. Estos son utilizados para controlar los niveles de acceso de un usuario al sistema. El paquete se relaciona con los subsistemas Presentación WSDL y Búsqueda de Usuario.

Subsistema Eliminar Nivel de Acceso (WS)

Este subsistema es un Servicio Web que es invocado para eliminar los niveles de acceso de los usuarios de la institución con este servicio se elimina totalmente los niveles de acceso de un usuario determinado o

sea no puede acceder a ningún lugar o servicio de la institución, este servicio depende del servicio Búsqueda de Usuario pues para eliminar los niveles de acceso de un usuario es necesario localizarlo primero en la base de datos.

Subsistema Modificar Nivel de Acceso (WS)

Este subsistema es un Servicio Web que es invocado para modificar los niveles de acceso de los usuarios de la institución con este servicio se modifican los niveles de acceso establecidos para un usuario determinado o sea agregarle o quitarle niveles de acceso, este servicio depende del servicio Búsqueda de Usuario pues para modificar los niveles de acceso de un usuario es necesario localizarlo primero en la base de datos.

Subsistema Registrar Acceso (WS)

Este subsistema es un Servicio Web, es utilizado por la institución para registrar los accesos de los usuarios a la misma, así como a las diferentes áreas a las que puede acceder un usuario determinado, este servicio depende del servicio Búsqueda de Usuario pues para ver si el usuario tiene acceso o no a la institución así como para ver el área a la que tiene acceso es necesario buscarlo primero en la base de datos.

Subsistema Solicitar Préstamo en Sala (WS)

Este subsistema es un Servicio Web, es utilizado para que el usuario pueda llenar la solicitud de préstamos de documentos o materiales que están disponibles en la institución, estos documentos o materiales pueden estar dentro o fuera de la institución, el usuario debe llenar un formulario con sus datos personales así como el tipo de documento o material que desea consultar.

Subsistemas de Búsqueda Documentos (WS)

Este subsistema es un Servicio Web que está disponible para que el usuario pueda buscar la documentación que está disponible en la red esta documentación le sirve para la preparación de las teleclases

Paquete Reproducción de Documentos

En este paquete se encuentran los subsistemas de Impresión y Fotocopia, Digitalización de Documentos y Transferencia de Formato, estos subsistemas tienen en común que se utilizan para llenar la solicitud de reproducir un documento con el objetivo de sacarlo de la institución y poder consultarlo fuera de ella.

Subsistema Impresión y Fotocopia

Este subsistema es un servicio que brinda la institución con el objetivo de que los usuarios puedan llenar la solicitud de imprimir o fotocopiar un documento para llevarlo a sus casas y para poderlo consultar posteriormente.

Subsistema Digitalización de Documentos

Este subsistema es un servicio que brinda la institución con el objetivo de que los usuarios puedan llenar la solicitud de digitalizar un documento que les sea necesario para la preparación de sus teleclases.

Subsistema Transferencia de Formato

Este subsistema es un servicio que brinda la institución con el objetivo de que los usuarios puedan llenar la solicitud de hacer una transferencia de formato esto se hace con el objetivo de que el usuario pueda extraer información que es demasiado grande y tiene que hacerlo en CD, DVD, etc.

Paquete Reservación

En este paquete se encuentran los subsistemas de Tiempo de Máquina, Local de Trabajo, Préstamo de PC Portátil, estos subsistemas tienen en común que son utilizados para que el usuario solicite reservaciones para poder realizar actividades en la institución, estas reservaciones se hacen de forma tal que ese usuario sea el único que este utilizando ese servicio en ese momento.

Subsistema Tiempo de Máquina

Este subsistema es un servicio que brinda la institución con el objetivo de que los usuarios puedan solicitar el tiempo de máquina que necesitan para su preparación o para la preparación de sus teleclases, este servicio debe brindar la posibilidad de que los usuarios conozcan las máquinas que están disponibles y las que no lo están, en que momento van a estarlo.

Subsistema Local de Trabajo

Este subsistema es un servicio que brinda la institución con el objetivo de que los usuarios puedan solicitar un local de trabajo para su preparación o para la preparación de sus teleclases, este servicio debe brindar la posibilidad de que los usuarios conozcan los locales de trabajo que están ocupados y los que no lo están.

Subsistema Préstamo de PC Portátil

Este subsistema es un servicio que brinda la institución con el objetivo de que los usuarios puedan solicitar el préstamo de una PC portátil en caso de que le sea necesario su uso, este servicio se brinda para que el usuario utilice la PC portátil dentro de la institución.

Subsistema Validación de Datos

Este subsistema es el encargado de validar los datos que llegan a través de los Servicios Web, es el subsistema que vincula a los Servicios Web con la capa de acceso a datos, en el se encuentran una serie de funciones que permiten la entrada o no de los datos en dependencia de si son esos o no los datos que se desean.

Subsistema Acceso a Datos Implementación (DAO)

En este subsistema se encuentran las funciones de la implementación del sistema, como tal es donde se encuentran todas las funciones que van a ser instanciadas por los Servicios Web.

Subsistema Acceso a Datos (ADODB)

En este subsistema se encuentran las funciones que tienen que ver con el acceso a las base de datos de la institución, ya sea para hacer solicitudes de información o para realizar una operación dentro de la base de datos.

Paquetes Arquitectónicos significativos de los Servicios Web

Vista General de la Estructura por Paquetes de Clases de los Servicios Web

Los subsistemas están compuestos por clases, las clases se agrupan por paquetes para que se tenga una mejor comprensión de las mismas y para que estén organizadas a la hora de trabajar con ellas. En el siguiente diagrama se muestra como se van a empaquetar las clases para los Servicios Web.

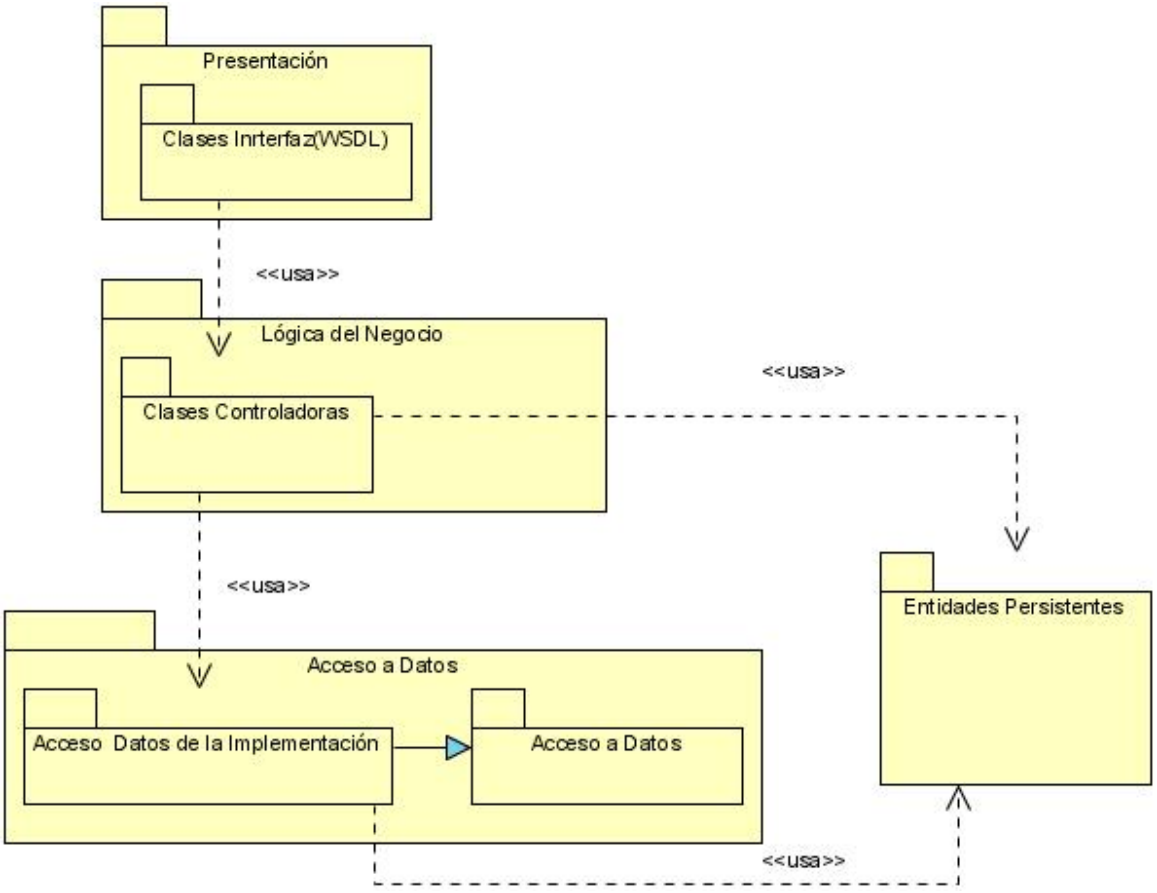

Fig.9. Vista General de la Estructura por Paquetes de los Servicios Web

En el Paquete de Presentación es donde van a estar las clases interfaz de los Servicios Web que brinda cada uno de los módulos, es donde van a estar registrados todos los Servicios Web que se exponen en el WSDL (Lenguaje de Descripción del Servicio Web). A continuación se muestra el diagrama de clases del Módulo Gestión de Usuario para este Paquete.


Paquete de Clases de la Interfaz del Servicio Web
Servicio_Web_Gestion_Persona
NuSOAP
Implementacion_Servicio_Web
delete_titulo($id_persona, $id_titulo)
delete_entrega_pc($id_persona, $id_entrega)
delete_idioma_domina($id_persona, $id_idioma)
delete_cursoUpT($id_persona, $id_curso, $id_rol)
delete_niveltv($id_persona, $id_nivel, $id_rol)
buscar_cantidad_grupo_edad()
buscar_cantidad_rol_tv($id)
buscar_cantidad_rol_upt($id)
buscar_cantidad_titulo_honorifico($id)
buscar_cantidad_categoria_doc($id)
buscar_cantidad_categoria_investigacion($id)
buscar_cantidad_grado_cientifico($id)
buscar_cantidad_nivel_academico($id)
buscar_cantidad_persona()
buscar_persona_con_pc()
buscar_persona_no_pc()
buscar_persona_centro_trabajo($organismo, $centr...
buscar_curso_upt($curso, $rol, $organismo)
buscar_nivel_tv($curso, $rol, $organismo)
buscar_persona_categoria_investigacion($id, $espe...
buscar_persona_titulo_honorifico($id, $especialidad...
buscar_persona_nivel_academico($id, $especialida...
buscar_persona_categoria_docente($id, $especialid...
buscar_persona_grado_cientifico($id_grado, $espec...
insertar_persona($key, $P_Nombre, $S_Nombre, $...
actualizar_persona($key, $Id_persona, $P_Nombre,...
buscar_persona_identificador($id)
eliminar_persona_identificador($key, $id)
busqueda_simple($nombre, $apellido1, $apellido2, ...
buscar_idioma($id_idioma, $lee, $escribe, $habla)
vaciar_bd_persona()
get_estructura_general($obj)
get_estructura_simple($obj)
devolver_nomencladores()
eliminar_cursoUpT($id)
actualizar_cursoUpT($id, $desc)
devolver_curso_upt($id)
devolver_nivel_tv($id)
eliminar_nivel_tv($id)
actualizar_nivel_tv($id, $desc)
devolver_especialidad($id)
actualizar_especialidad($id, $esp)
eliminar_especialidad($id)
eliminar_cargo($id)
devolver_cargo($id)
actualizar_cargo($id, $desc)
get_array_nivel_tv()
get_array_curso_upt()
get_array_rol()
get_array_idioma()
get_array_titulo()
get_array_categoria_doc()
get_array_grado_cientifico()
get_array_categoria_inv()
get_array_especialidad()
get_array_nivel_academico()
get_array_cargo()
get_array_organismo()
get_array_color_ojos()
get_array_estado_civil()
get_array_color_piel()
get_array_sexo()

Fig.10. Clases del Paquete Presentación

En el Paquete Lógica del Negocio que donde va estar la funcionalidad de los Servicios Web, contiene un grupo de clases controladoras de las entidades del negocio estas clases permiten una serie de funciones fundamentales para el negocio. A continuación se muestra el diagrama de clases del Módulo Gestión de Usuario para este Paquete.

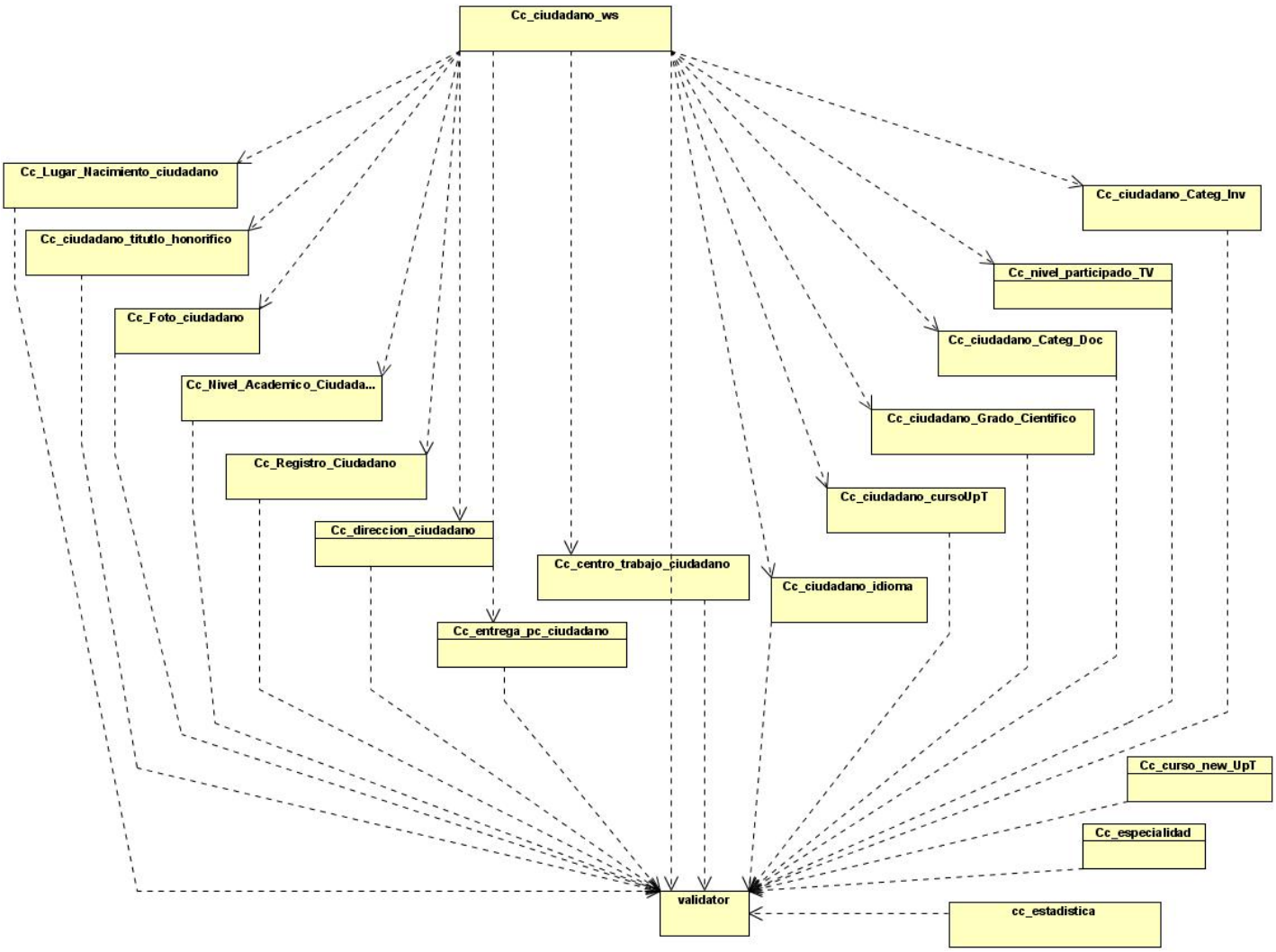

Fig.11. – Clases del Paquete Lógica de Negocio

En el Paquete de Acceso a Datos están las clases que posibilitan la persistencia y recuperación de objetos, este paquete a la vez esta divido en dos paquetes más pequeños los cuales son el Paquete de Acceso a Datos de la Implementación, donde se encuentran las clases encargadas de manejar la persistencia de los datos en las diferentes Base de Datos. El otro paquete es el Paquete de Acceso a Datos que es utilizado por el Paquete de Acceso a Datos de Implementación, en este paquete es donde se encuentran un conjunto de clases que permiten la conexión a las diferentes Bases de Datos. Este paquete permite a la aplicación abstraerse del origen de los datos y de la lógica de su persistencia logrando un bajo acoplamiento entre sus componentes. A continuación se muestra el diagrama de clases del Módulo Gestión de Usuario para este Paquete.

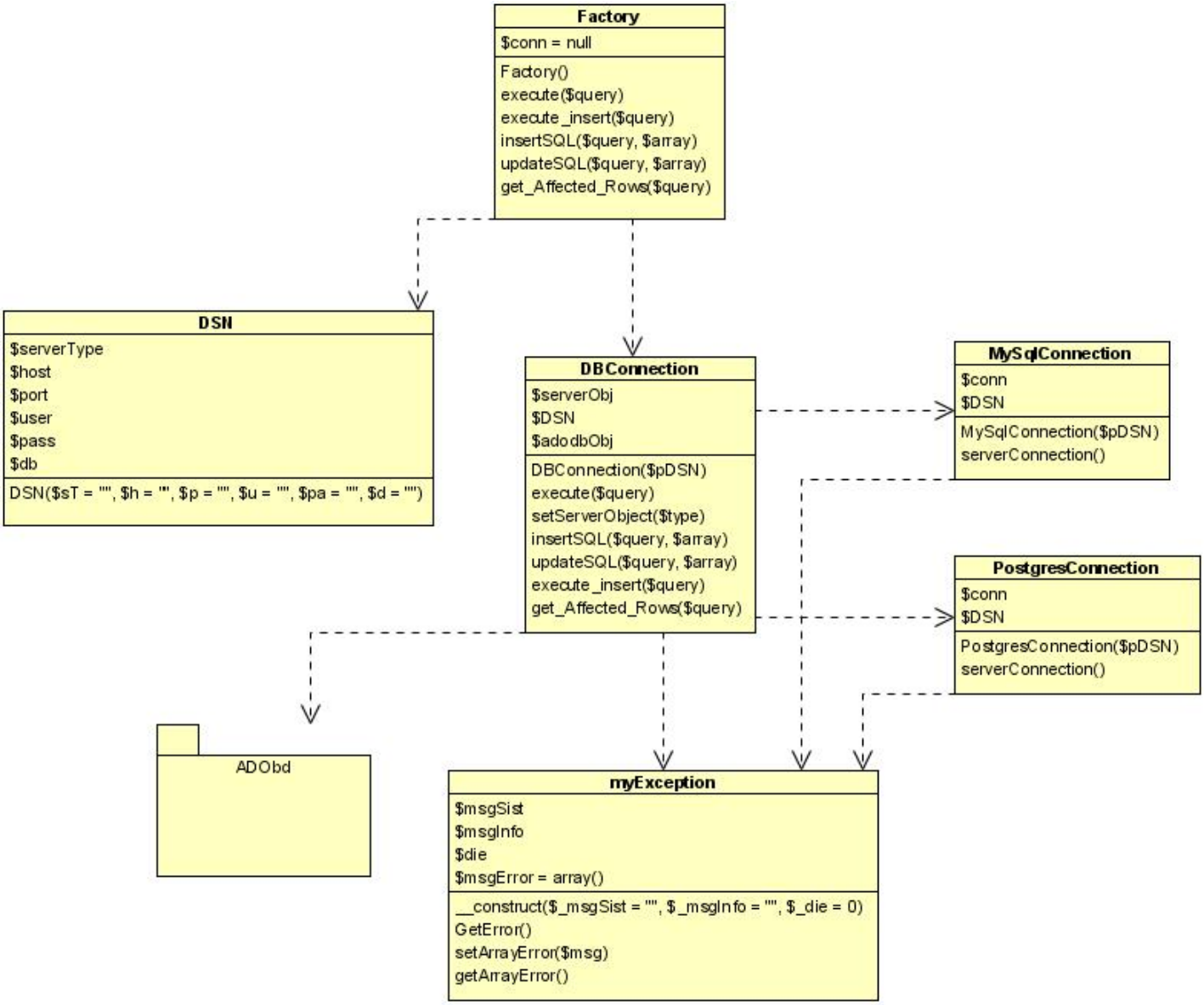

Fig.12. – Clases de Acceso a Datos del Paquete Acceso a Datos

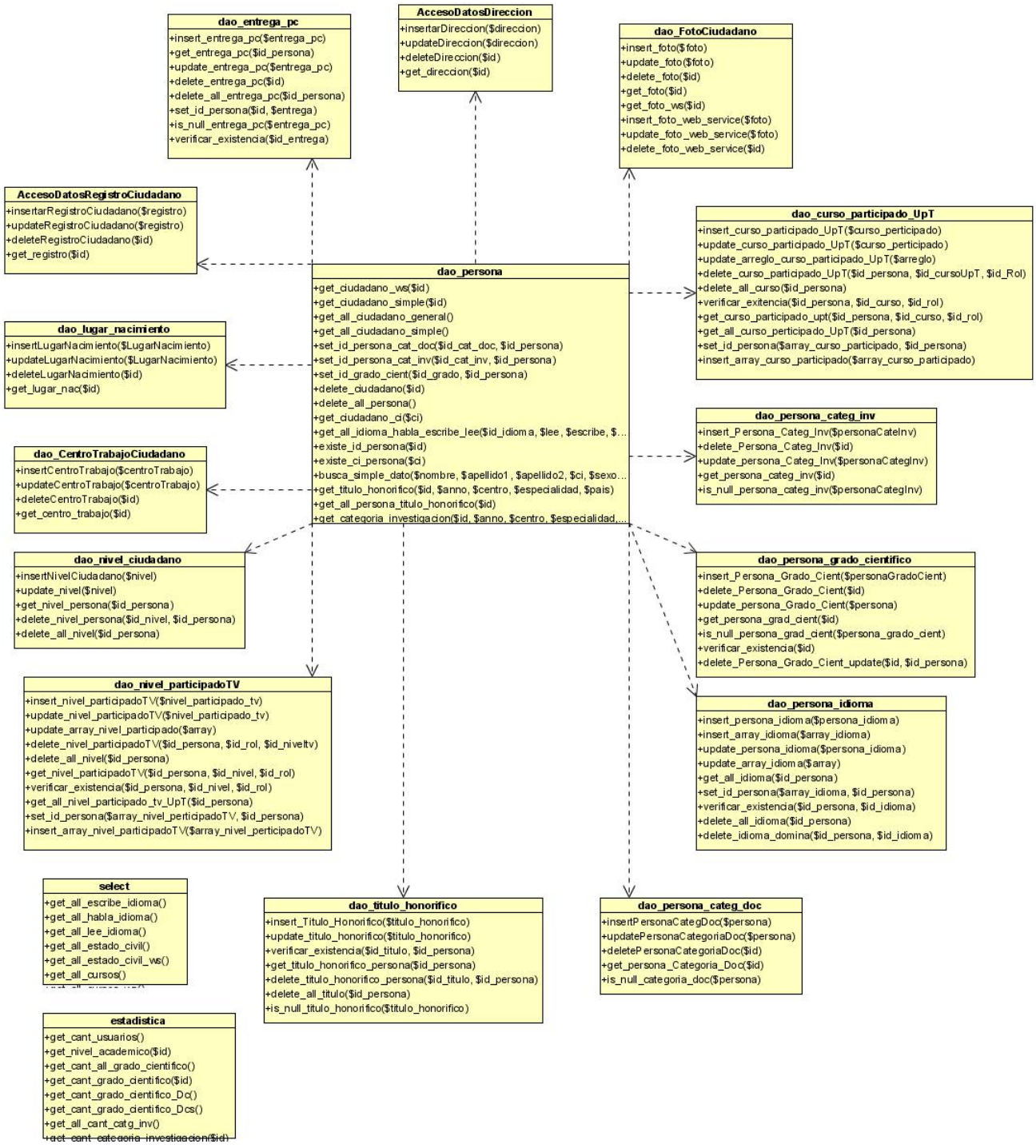

Fig.13. – Clases de Acceso a Datos de la Implementación del Paquete Acceso a Datos

En el Paquete Entidades Persistentes existen clases que no tienen comportamiento solo propiedades y son representaciones de entidades reales del dominio, estas clases son la mayoría persistentes y son accedidas por el Paquete Lógica del Negocio y el Paquete Acceso a Datos. A continuación se muestra el diagrama de clases del Módulo Gestión de Usuario para este Paquete.

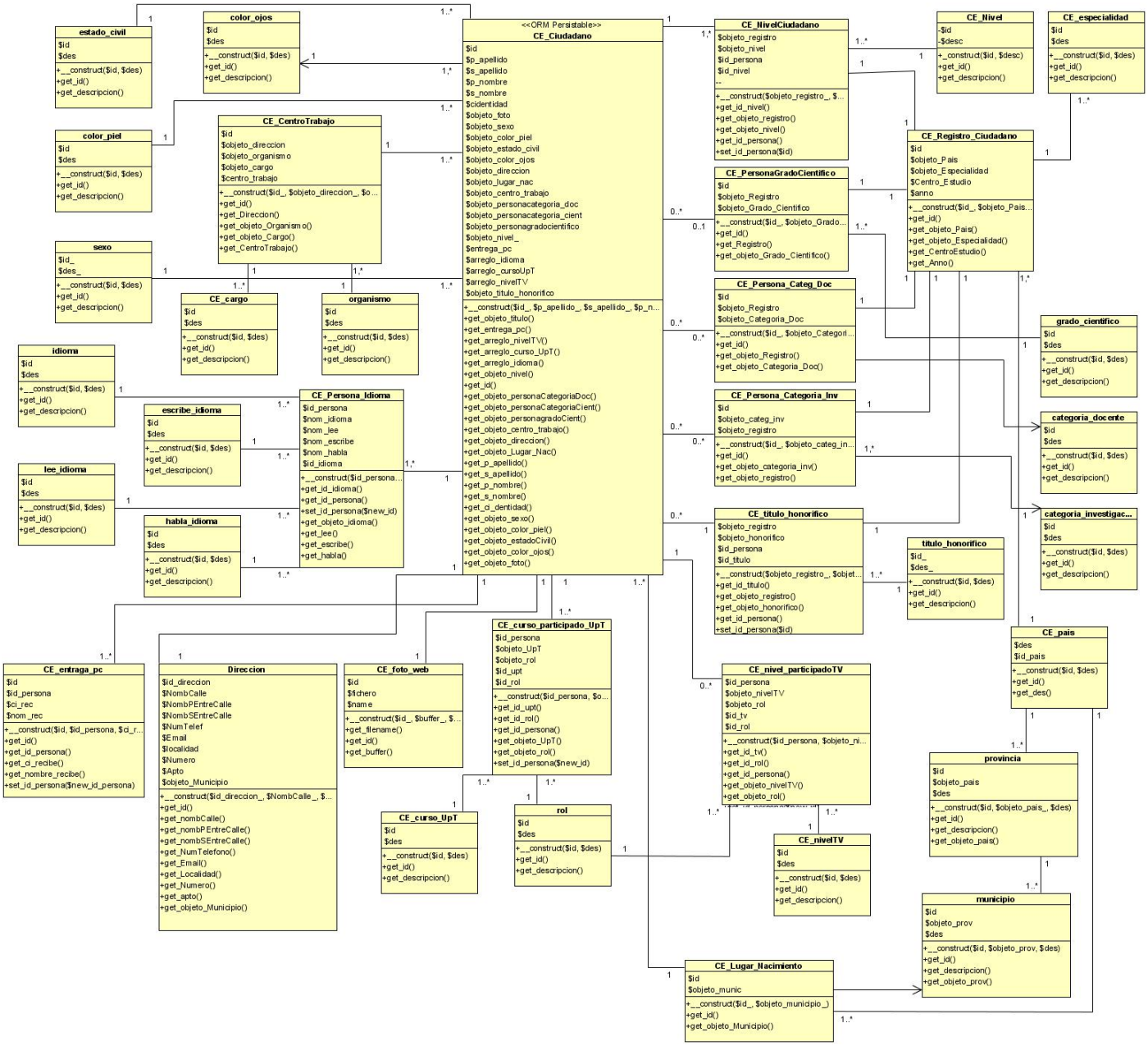

Fig.14. – Clases del paquete Entidades Persistentes

Vista de Procesos

En esta vista se van a agrupar los procesos por módulos y como interactúan los mismos en cada uno de los módulos, para esto se escogieron los procesos más importantes en cada módulo. En este caso por lo general cada proceso representa un Servicio Web a implementar.

Para el Modulo Gestión de Usuarios se identificaron los siguientes procesos Gestionar Usuario, Eliminar Usuario, Modificar Usuario, Adicionar Usuario y Buscar Usuario.

Gestionar Usuario: Este proceso es el que permite gestionar todo lo relacionado con la gestión de usuarios de la institución para gestionar esta información se apoya en los procesos, Buscar Usuario, Eliminar Usuario, Adicionar Usuario y Modificar Usuario que constituyen al mismo tiempo Servicios Web.

Buscar Usuario: Es un proceso que constituye un Servicio Web, se utiliza para buscar un usuario en la base de datos.

Eliminar Usuario: Es un proceso que constituye un Servicio Web, permite eliminar un usuario que este previamente registrado como miembro de la institución.

Agregar Usuario: Es un proceso que constituye un Servicio Web, permite agregar un nuevo usuario a la base de datos de la institución.

Modificar Usuario: Es un proceso que constituye un Servicio Web, permite modificar los datos personales de un usuario determinado que pertenezca a la institución.

Para el Modulo Acceso se identificaron los siguientes procesos Gestionar Niveles de Acceso, Solicitar Acceso, Eliminar Nivel de Acceso, Modificar Nivel de Acceso, Asignar Nivel de Acceso, Buscar Usuario.

Gestionar Niveles de Acceso: Este proceso es el que permite gestionar todo lo relacionado con los niveles de acceso de los usuarios de la institución para ello se apoya en los procesos, Eliminar Nivel de

Acceso, Modificar Nivel de Acceso, Asignar Nivel de Acceso y Buscar Usuario que constituyen al mismo tiempo Servicios Web.

Buscar Usuario: Este proceso ya esta explicado en el Modulo Gestión de Usuario
Eliminar Nivel de Acceso: Es un proceso que constituye un Servicio Web, permite eliminar por completo los niveles de acceso de un usuario determinado.

Asignar Nivel de Acceso: Es un proceso que constituye un Servicio Web, permite asignar niveles de acceso a usuarios que han sido agregados a la base de datos de la institución.

Modificar Nivel de Acceso: Es un Proceso que constituye un Servicio Web, permite modificar los niveles de acceso de un usuario determinado o sea agregarle o quitarle niveles de acceso.

Solicitar Acceso: Este proceso es el que permite a los usuarios solicitar el acceso a la institución o a un área determinada de la misma, para ello se apoya en el proceso de Buscar Usuario

Buscar Usuario: Este proceso ya esta explicado en el Modulo Gestión de Usuario
Trazabilidad del Paquete de Clases al Paquete de Componentes
Indica la trazabilidad entre los paquetes de clases de diseño y los paquetes de componentes de la implementación, o sea cada paquete de clases con su correspondiente paquete de componentes.

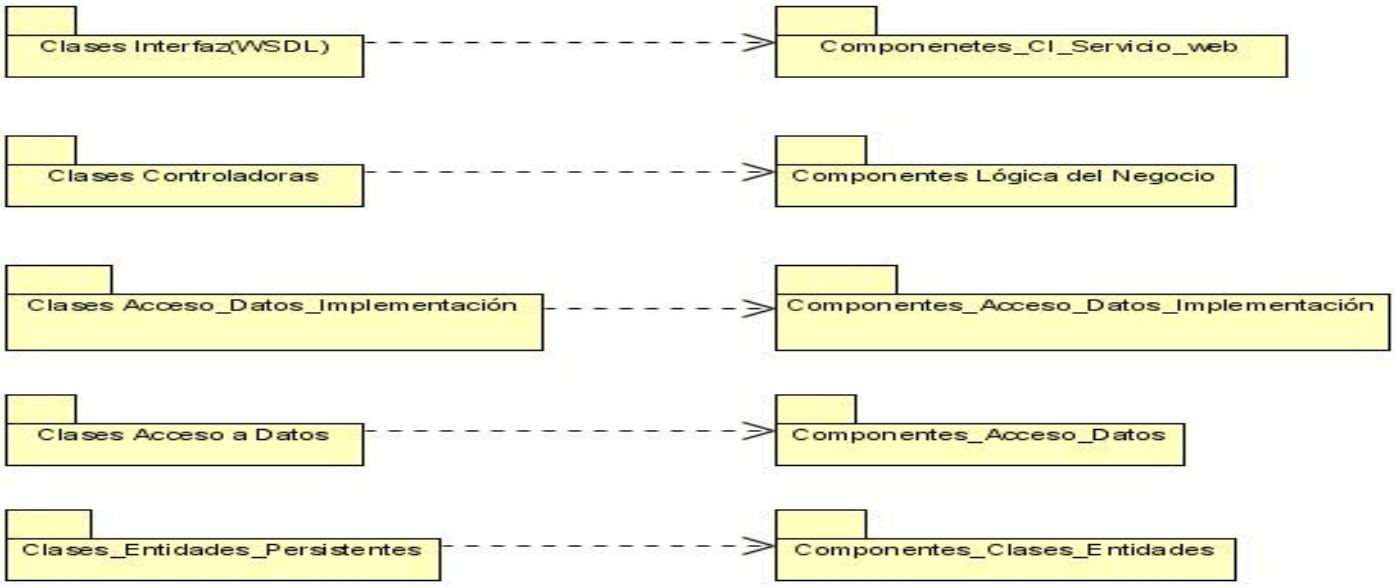

Fig.15. – Trazabilidad del paquete de clases al paquete de componentes

Vista de Implementación

Vista General de la Estructura por Paquetes de los Componentes de los Servicios Web

Los subsistemas de implementación están compuestos por componentes, los componentes se agrupan por paquetes para que se tenga una mejor comprensión de los mismos y para que estén organizados a la hora de trabajar con ellos, en el siguiente diagrama se muestra como se van a empaquetar los componentes para los Servicios Web.

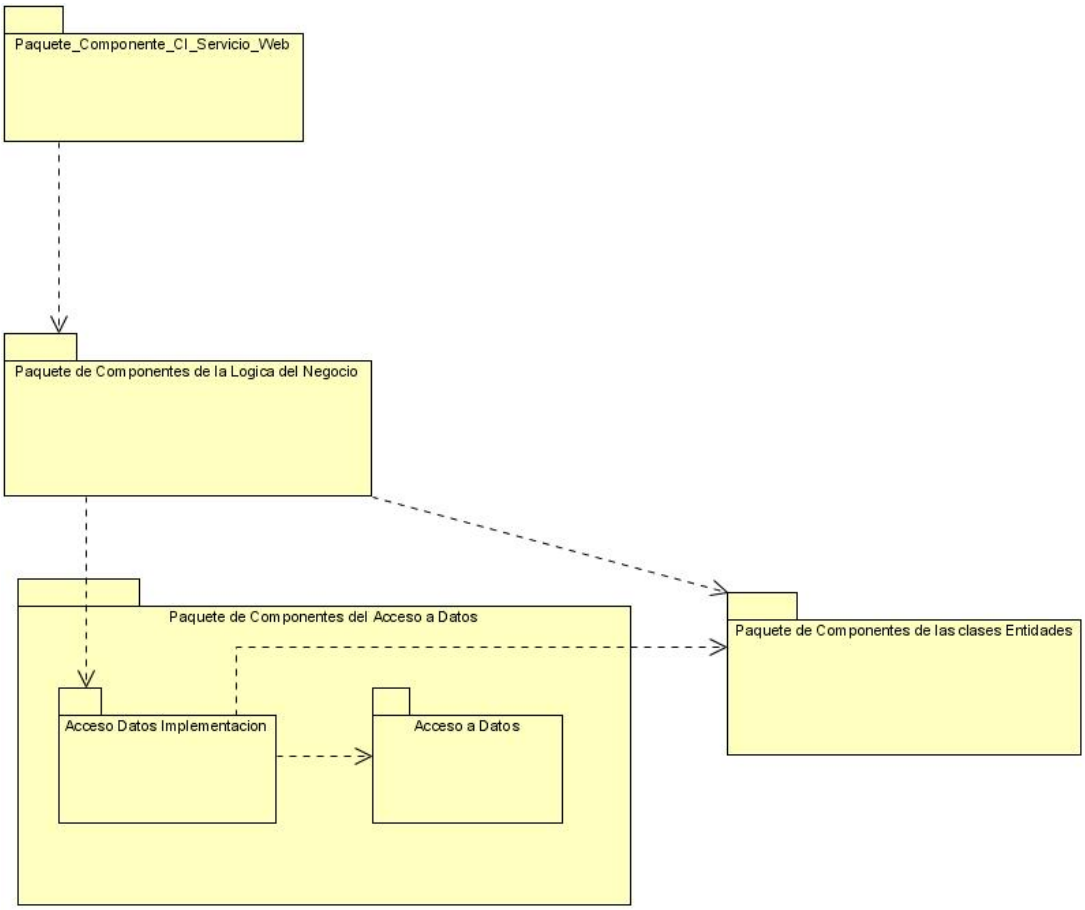

Fig.16. Vista General de la Estructura de Componentes por Paquetes de los Servicios Web

En el Paquete de Presentación es donde están los componentes que se van a implementar en la interfaz de los Servicios Web que brinda cada uno de los módulos, es donde van a estar registrados todos los Servicios Web que se exponen en el WSDL (Lenguaje de Descripción del Servicio Web). A continuación se muestra el diagrama de Componentes del Módulo Gestión de Usuario para este Paquete.

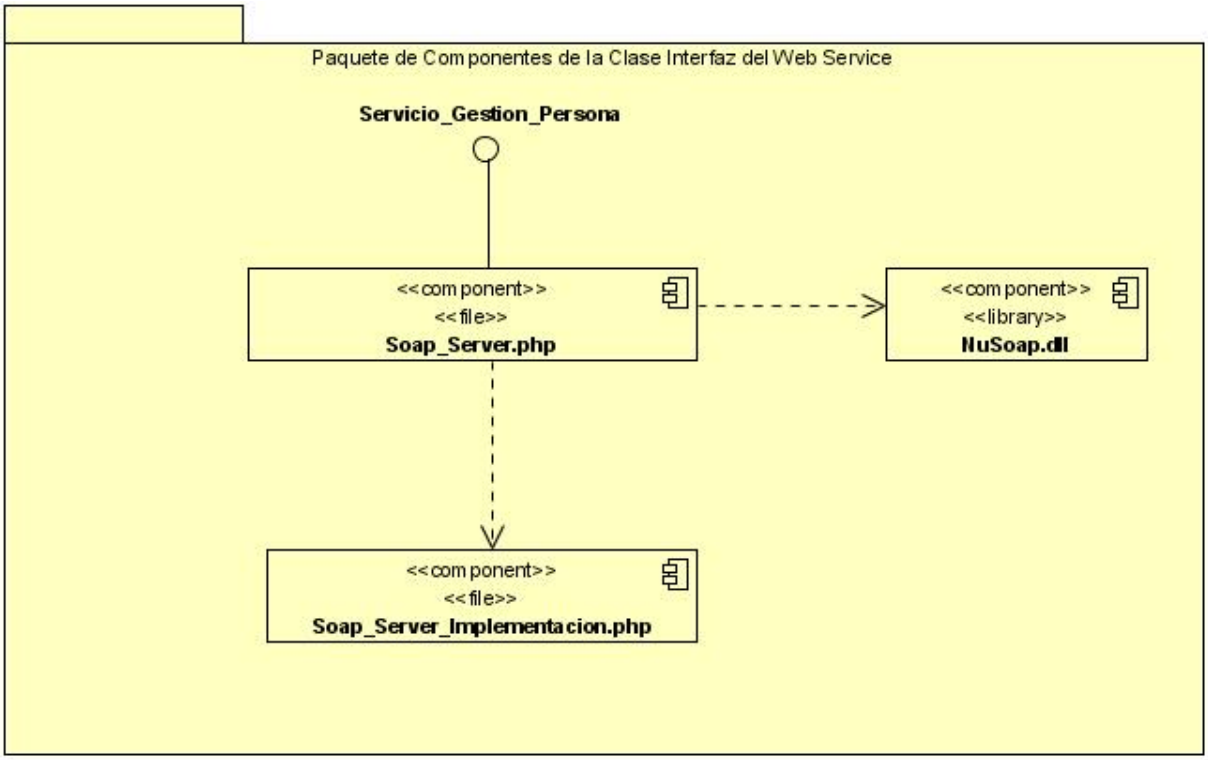

Fig.17. Diagrama de Componentes del Paquete Clase Interfaz del Servicio Web

En el Paquete Lógica del Negocio que donde esta la funcionalidad de los Servicios Web, contiene un grupo de componentes que se implementan con el objetivo de que tenga lugar la lógica del negocio estos componentes permiten una serie de funciones fundamentales para el negocio. A continuación se muestra el diagrama de Componentes del Módulo Gestión de Usuario para este Paquete.

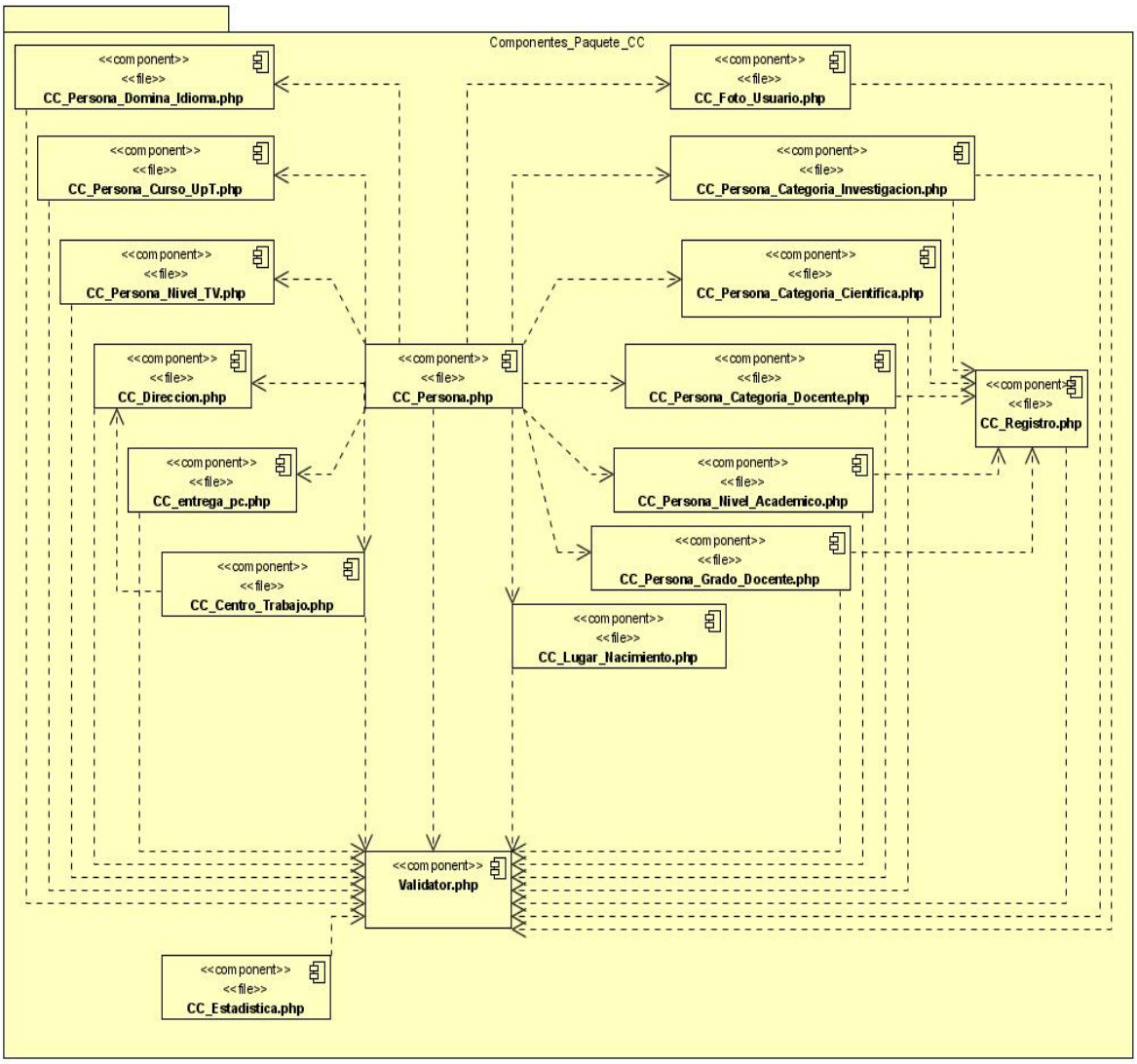

Fig.18. Diagrama de Componentes del Paquete Lógica del Negocio del Servicio Web

En el Paquete de Acceso a Datos están los componentes que posibilitan la persistencia y recuperación de objetos, este paquete a la vez esta divido en dos paquetes más pequeños los cuales son el Paquete de Acceso a Datos de la Implementación, donde se encuentran los componentes encargados de manejar la persistencia de los datos en las diferentes Base de Datos, estos componentes son de tipo Data Acces Object (DAO). El otro paquete es el Paquete de Acceso a Datos que es utilizado por el Paquete de Acceso a Datos de Implementación, en este paquete es donde se encuentran un conjunto de componentes que

permiten la conexión a las diferentes Base de Datos, estos componentes son ficheros y entre ellos esta DB_Conection.php, Factory.php, ADOdb.dll, etc. Este paquete permite a la aplicación abstraerse del origen de los datos y de la lógica de su persistencia logrando un bajo acoplamiento entre sus componentes. A continuación se muestra el diagrama de Componentes del Módulo Gestión de Usuario para este Paquete:

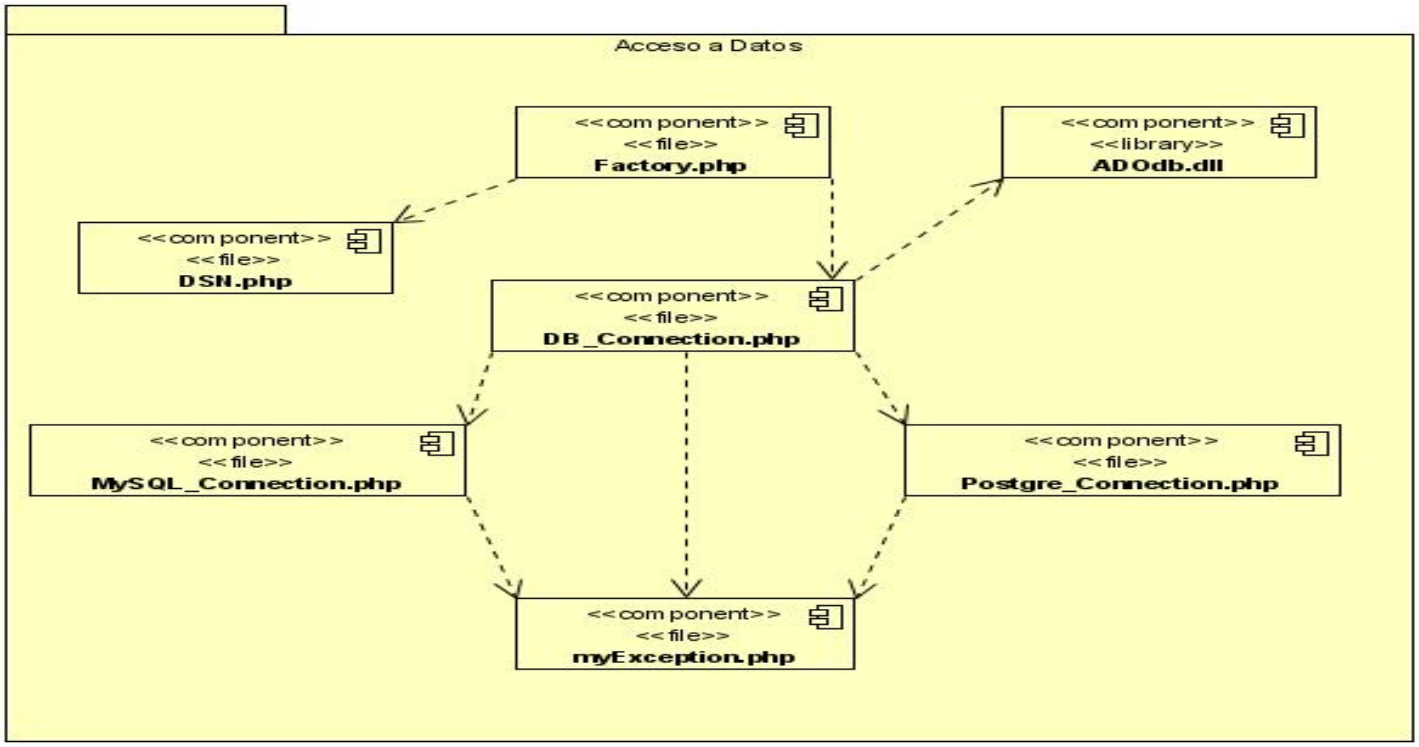

Fig.19. Diagrama de Componentes para el Paquete Acceso a Datos

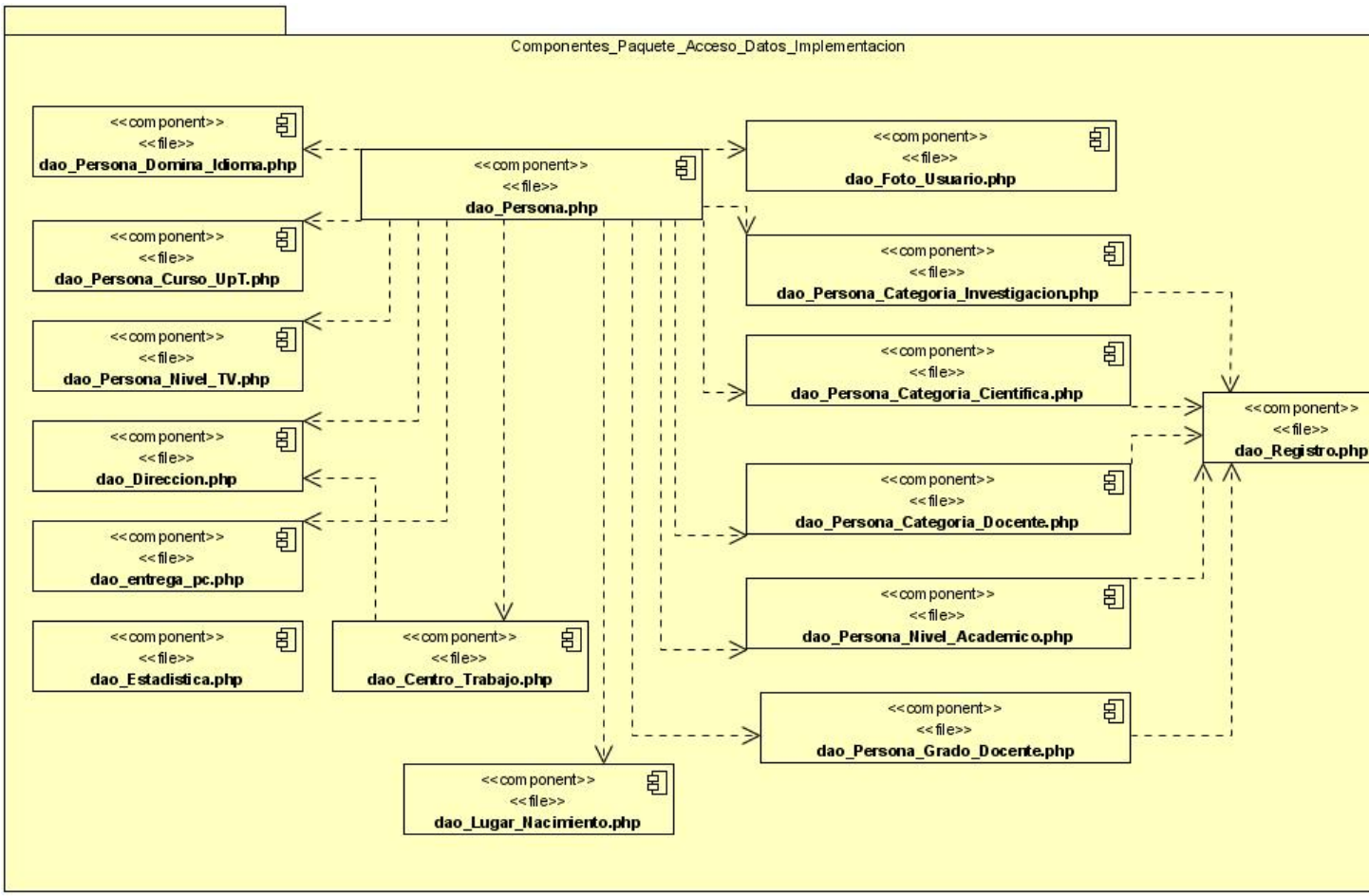

Fig.20. Diagrama de Componentes para el Paquete Acceso a Datos de la Implementación.

En el Paquete Entidades Persistentes existen componentes que no tienen comportamiento solo propiedades y son representaciones de entidades reales del dominio, estas componentes son la mayoría persistente y son accedidas por el Paquete Lógica del Negocio y el Paquete Acceso a Datos. A continuación se muestra el diagrama de Componentes del Módulo Gestión de Usuario para este Paquete

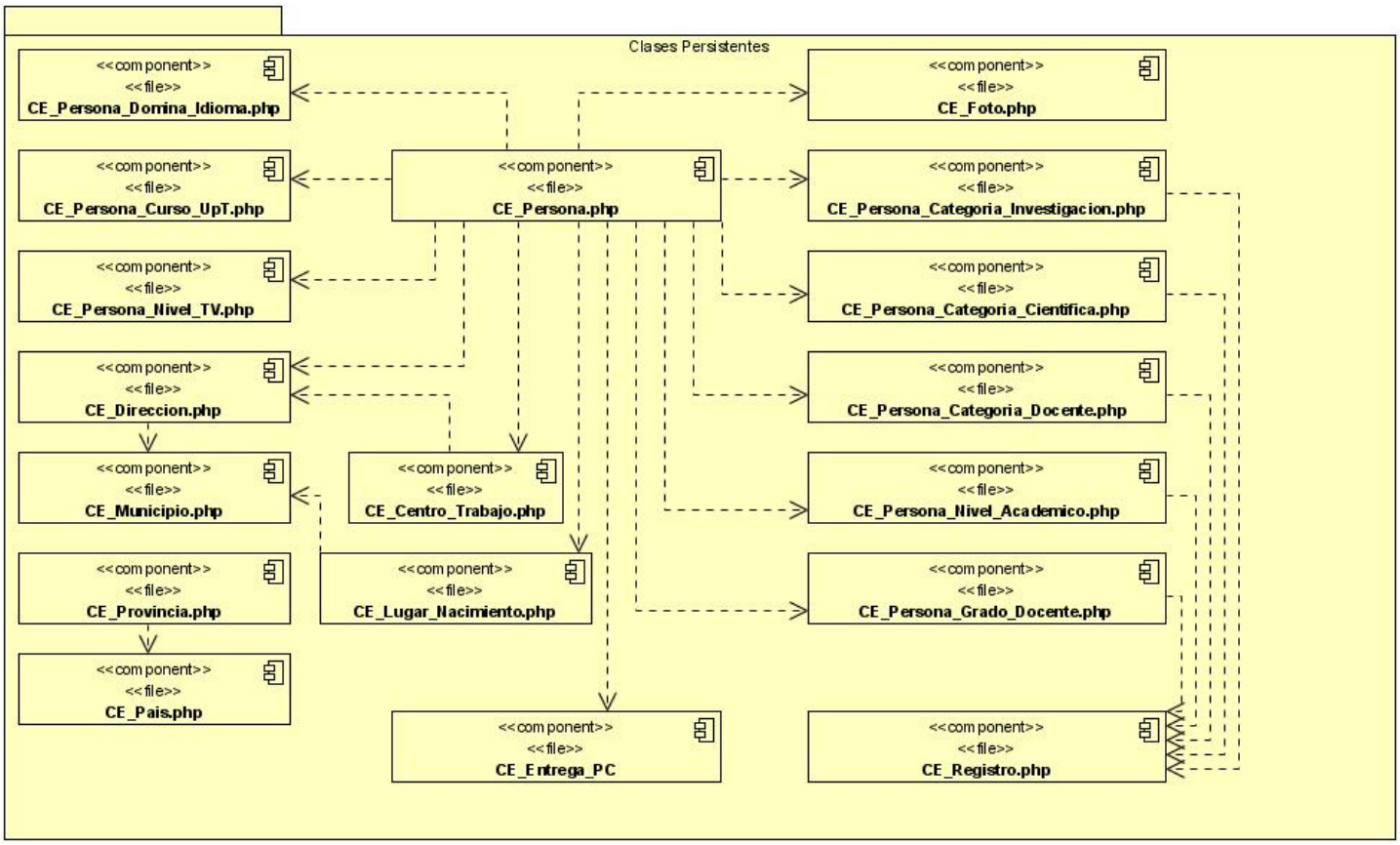

Fig.21. Diagrama de Componentes para el Paquete de Clases Persistentes

Vista de Despliegue

En esta vista se representan los nodos o sea los servidores en los que se van a guardar los datos de la institución así como cada uno de los Subsistemas de Implementación que está en dichos nodos. Los Servicios Web que fueron presentados como subsistemas de diseño y aquí son representados como Subsistemas de Implementación, es debido a que como cada no de ellos representan funcionalidades diferentes, o sea, como la implementación de un Servicio Web no depende de la implementación de otro se decidió tratarlos a cada uno de ellos como Subsistemas de Implementación.

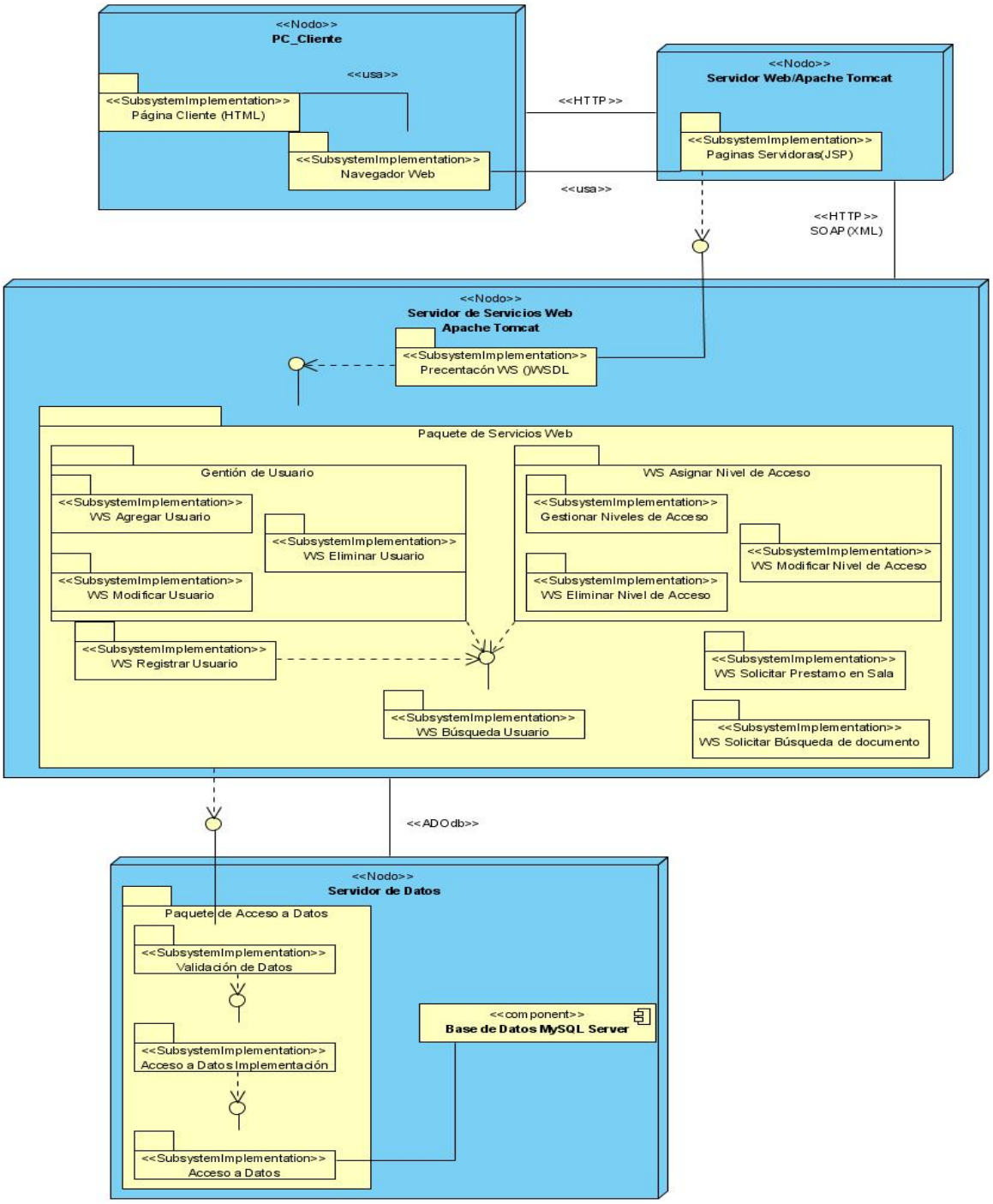

Fig.22. Diagrama de distribución

Nodo PC_Cliente

Es donde van a trabajar los clientes o sea los profesores que van a acceder a los servicios que brinda la institución, así como los trabajadores de la institución, en este nodo se encuentran los subsistemas de implementación que interactúan con los clientes estos subsistemas son las páginas servidoras y el Navegador Web.

Nodo Servidor Web

En este nodo que es un servidor Apache es donde se encuentran las páginas servidoras de la aplicación que están elaboradas en el CMS Infoglue por eso es que son .JSP o sea que son páginas hechas en Java, en este servidor es donde va a estar publicado el sitio de la institución.

Nodo Servidor de Servicios Web

En este nodo que es un servidor Apache es donde se encuentran todos los subsistemas de implementación que son a la vez Servicios Web que va a brindar la institución.

Paquetes de Servicios Web

Este paquete se encuentra en el nodo Servidor de Servicios Web es un paquete que se utilizó para organizar mejor el trabajo en el se encuentran todos los Servicios Web de implementación.

Subsistema de Implementación Búsqueda de Usuario

Este Subsistema de Implementación es un Servicio Web, este Servicio Web brinda la posibilidad de buscar usuarios en la base de datos de la institución, se le pasa un id y el busca esa persona, es uno de los servicios más importantes debido a que de él dependen otros servicios que se brindan en la institución.

Paquete de Gestión de Usuarios

Este paquete esta conformado por tres subsistemas de implementación que constituyen a la vez tres Servicios Web los mismos son, Subsistema de Implementación Eliminar Usuario, Subsistema de Implementación Agregar Usuario y el Subsistema de Implementación Modificar Usuario, estos subsistemas tienen en común que en ellos se implementan componentes que son utilizados para manejar la información con respecto a un usuario determinado de la institución.

Subsistema de Implementación Eliminar Usuario

Subsistema de Implementación que es un Servicio Web, este servicio se utiliza para eliminar usuarios que están registrados como miembros de la institución este servicio depende del servicio Búsqueda de Usuario ya que para eliminar a un usuario es necesario buscarlo primero.

Subsistema de Implementación Agregar Usuario

Subsistema de Implementación que es un Servicio Web, el mismo es utilizado para agregar un nuevo usuario a la institución con todos los datos personales del mismo, este servicio depende del servicio Búsqueda de Usuario ya que para agregar un nuevo usuario se debe comprobar si el mismo se encuentra o no en la base de datos de la institución.

Subsistema de Implementación Modificar Usuario

Subsistema de Implementación que es un Servicio Web, el mismo es utilizado para modificar los datos personales de un usuario determinado que ya este registrado en la Base de Datos de la institución este servicio depende del servicio Búsqueda de Usuario pues para modificar los datos personales de un usuario es necesario buscarlo primero en la base de datos.

Paquete Gestionar Niveles de Acceso

Este paquete está conformado por tres Subsistemas de Implementación que son a la vez tres Servicios Web y los mismos son, Subsistema de Implementación Asignar Nivel de Acceso, Subsistema de Implementación Eliminar Nivel de Acceso, Subsistema de Implementación Modificar Nivel de Acceso, estos subsistemas tienen en común que en ellos se implementan componentes que son utilizados para controlar los niveles de acceso de un usuario determinado de la institución.

Subsistema de Implementación Asignar Nivel de Acceso

Este Subsistema de Implementación es un Servicio Web que es invocado para asignarle diferentes niveles de acceso a los usuarios de la institución y así establecer los lugares y servicios a los que va a tener privilegios un usuario determinado, este servicio depende del servicio Búsqueda de Usuario pues para asignarle los niveles de acceso a un usuario es necesario buscarlo primero en la base de datos.

Subsistema de Implementación Eliminar Nivel de Acceso

Este Subsistema de Implementación es un Servicio Web que es invocado para eliminar los niveles de acceso de un determinado usuario de la institución, con este servicio se eliminan totalmente los niveles de acceso del usuario, o sea no puede acceder a ningún lugar o servicio de la institución, este servicio depende del servicio Búsqueda de Usuario pues para eliminar los niveles de acceso de un usuario es necesario localizarlo primero en la base de datos.

Subsistema de Implementación Modificar Nivel de Acceso

Este Subsistema de Implementación es un Servicio Web que es invocado para modificar los niveles de acceso de un determinado usuario de la instituciónn con este servicio se modifican los niveles de acceso establecidos para el usuario, o sea agregarle o quitarle niveles de acceso, este servicio depende del servicio Búsqueda de Usuario pues para modificar los niveles de acceso de un usuario es necesario localizarlo primero en la base de datos.

Subsistema de Implementación Registrar Acceso

Este Subsistema de Implementación es un Servicio Web, es utilizado por la institución para registrar los accesos de los usuarios a la misma, así como a las diferentes áreas a las que puede acceder un usuario determinado, este servicio depende del servicio Búsqueda de Usuario pues para ver si el usuario tiene acceso o no a la institución así como para ver el área a la que tiene acceso es necesario buscarlo primero en la base de datos.

Subsistema de Implementación Solicitar Préstamo en Sala

Este Subsistema de Implementación es un Servicio Web, es utilizado para que el usuario pueda llenar la solicitud de préstamos de documentos o materiales que están disponibles en la institución, estos documentos o materiales pueden estar dentro o fuera de la institución, el usuario debe llenar un formulario con sus datos personales así como el tipo de documento o material que desea consultar.

Subsistemas de Implementación Búsqueda Documentos

Este Subsistema de Implementación es un Servicio Web que está disponible para que el usuario pueda buscar la documentación que está disponible en la red, esta documentación le sirve para la preparación de las teleclases.

Nodo Servidor de Datos

En este nodo se encuentran las Base de Datos de la institución en, esta base de datos es donde va a estar guardada toda la información que se desea manejar por parte de la institución, dichas Bases de Datos son MySQL Server, y en ellas también se encuentran los Subsistemas de Implementación que tienen que ver con los accesos a los datos.

Paquete de Acceso a Datos

En este paquete que se encuentra en el nodo de acceso a datos van a estar los subsistemas de Implementación que tienen que ver con los accesos a los datos de las Bases de Datos, este paquete se utilizó para organizar mejor el trabajo.

Subsistema de Implementación Validación de Datos

Este Subsistema de Implementación es el encargado de validar los datos que llegan a través de los Servicios Web, es el subsistema que vincula a los Servicios Web con la capa de acceso a datos en el se encuentran una serie de funciones que permiten la entrada o no de los datos en dependencia de si son esos o no los datos que se desean.

Subsistema de Implementación Acceso a Datos (DAO)

En este Subsistema de Implementación se encuentran las funciones de la implementación del sistema, como tal es donde se encuentran todas las funciones que van a ser instanciadas por los Servicios Web.

Subsistema de Implementación Acceso a Datos (ADOdb)

En este Subsistema de Implementación se encuentran las funciones que tienen que ver con el acceso a las bases de datos de la institución, ya sea para hacer solicitudes de información o para realizar una operación dentro de la base de datos.

Base de Datos MySQL Server

Este es un componente que representa la Base de Datos que tiene relaciones con los subsistemas de acceso a datos.

CAPITULO 3: EVALUACIÓN DE LA ARQUITECTURA

Evaluación de la Arquitectura de Software

Uno de los factores que determina el éxito o fracaso de un sistema de software, es su arquitectura. Con esto nos referimos a la estructura o conjunto de estructuras del sistema, las cuales están compuestas de elementos de software, propiedades externas visibles de estos elementos y las relaciones entre sí. El diseñar debidamente una arquitectura de software, garantiza que el sistema de software cumpla con uno o varios atributos de calidad. Por ejemplo, que sea fácil de usar, confiable o seguro. Sin embargo si la arquitectura no se diseña de forma apropiada, el sistema de software resultante no logrará sus objetivos. De nada sirve un sistema de software que no cumple con los tiempos de respuesta requeridos por el cliente, o que es complejo de modificar, difícil de usar o vulnerable a ataques.
Por lo regular, no se conoce sino hasta el final del desarrollo del sistema de software, si éste cumplió o no con los atributos de calidad que se especificaron en los requerimientos no funcionales. Dicho conocimiento tardío, implica tomar demasiados riesgos innecesarios, un ejemplo es, descubrir fallas en el sistema de software debido a que en la fase de diseño no se eligió apropiadamente una arquitectura. Para reducir tales riesgos y, como una buena práctica de ingeniería, es recomendable llevar acabo evaluaciones a la arquitectura.

¿Cómo determinamos que forma parte de una Arquitectura?

Debe ser un componente, relación entre componentes, o una propiedad (de componentes o relaciones) que necesita ser externamente visible, con el objetivo de razonar sobre la habilidad del sistema de alcanzar sus requerimientos de calidad, o de soportar la descomposición del sistema en partes independientemente implementables.

¿Cómo puedo estar seguro que la arquitectura elegida es la correcta para mi software?

Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar las decisiones arquitectónicas con respecto a su impacto sobre dichos atributos.

¿Por qué es necesario evaluar una arquitectura de software?

El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe señalar que los requerimientos no funcionales, también son conocidos como atributos de calidad. De acuerdo al estándar IEEE 6 10.12-1990, un atributo de calidad es una característica que afecta la calidad de un elemento. En esta definición el término “característica” se refiere a aspectos no funcionales, mientras que el término “elemento” se refiere a un componente o sistema.

Los atributos de calidad se clasifican en dos grupos: operacionales y de desarrollo. Los atributos operacionales son las cualidades del sistema que están en operación, por ejemplo: rendimiento, confiabilidad, disponibilidad, tolerancia a fallas. En cambio, los atributos de desarrollo son las cualidades del sistema que son relevantes desde una perspectiva del desarrollo de software, por ejemplo: facilidad de modificación, facilidad de re-utilización, flexibilidad.

Evaluar es la manera mas económica para prevenir y evitar todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es la arquitectura diseñada para el sistema. Una evaluación de la arquitectura no te da un si o un no, si es buena o mala, o una calificación. Esta te dice donde está el riesgo, es decir fortalezas y debilidades identificadas de la arquitectura. Después de una evaluación se pueden tomar algunas decisiones como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación o si hay que reforzar la arquitectura o si hay que comenzarla toda de nuevo. Por todo lo visto se puede ver que cuando mas temprano se encuentre un problema en un proyecto del software, mejor.

¿Qué avaluamos en una Arquitectura del Software?

La actividad más compleja durante el desarrollo del proyecto del software es, transformar las especificaciones de los requerimientos de calidad (Quality Requerimients, QRs) en una AS.

El objetivo es evaluar el potencial del diseño de la arquitectura para alcanzar los niveles requeridos por los QRs.

Podemos categorizar los QRs en dos, de desarrollo y operacionales.

Los QRs de desarrollo son cualidades del sistema que son relevantes para la ingeniería de software, por ejemplo, maintainability, reusability, flexibility and demostrability.
Los QRs operacionales, son cualidades del sistema en operación, por ejemplo, performance, reliability, robustness and fault-tolerance.

Primeros inicios

Los primeros esfuerzos en realizar evaluaciones a arquitecturas de software utilizando un proceso estructurado y definido, fueron descritos en un trabajo seminal hecho por Parnas y Weiss en 1985. En él se propone utilizar revisiones de diseño activas, que consisten en detectar errores e inconsistencias, por ejemplo aquellos que no fueron detectados en la fase de requerimientos. Este proceso consiste en elaborar una serie de cuestionarios cuidadosamente escritos, de tal manera que el revisor no pueda responderlos de una manera pasiva, es decir, con un “si” o un “no”.
Cada cuestionario se diseña para encontrar diferentes tipos de errores, por lo que cada uno, evalúa aspectos específicos del diseño. Estos, a su vez, junto con la documentación del diseño, se entregan para su evaluación a un grupo de revisores expertos en uno o varios aspectos específicos del diseño. En promedio, la duración de dichas revisiones se lleva de uno a dos días, dependiendo de la complejidad del diseño, así como del número de revisores.

¿Cuándo es recomendable evaluar?

La evaluación a la arquitectura puede efectuarse en varios momentos, dependiendo de la etapa de construcción en que se encuentre. La evaluación clásica se efectúa una vez que la arquitectura se ha terminado y aún no se implementa. La evaluación temprana se lleva a cabo una, o varias veces durante la etapa de construcción de la arquitectura, mientras que la evaluación tardía, se realiza cuando la arquitectura existe y la implantación se ha completado. Clements, Kazman y Klein proponen dos reglas de oro para determinar el momento de efectuar la evaluación:
Realizar una evaluación cuando el equipo de desarrollo comience a tomar decisiones que dependan de la arquitectura.
Cuando el costo de no tomar estas, podría pesar más, que el costo de realizar una evaluación.

¿Quiénes participan en la evaluación?

Generalmente las evaluaciones a la arquitectura se hacen por los miembros del equipo del desarrollo, tales como el arquitecto, diseñador y administrador del proyecto. Sin embargo, puede haber excepciones en las que se contrate a un grupo de personas especialistas para realizar la evaluación. El cliente también se interesa por los resultados obtenidos de la evaluación, ya que puede decidir continuar o no con el proyecto, dependiendo de los resultados. Un ejemplo de este caso es cuando tras haber efectuado una evaluación temprana se determina que no es posible implantar la arquitectura con la infraestructura disponible, ya que esta no cumplirá con los tiempos de respuesta requeridos por el cliente.

¿Quiénes están involucrados?

Equipo de evaluación: Es el personal que conducirá la evaluación y realizara el análisis.
Stakeholders: Ese grupo de personas y organizaciones que tienen un interés en el desarrollo del proyecto (por ej. accionistas, clientes, proveedores, competidores, inversionistas, grupos de comercio, etc.).

Interesados externos (Organización cliente, usuarios finales, administradores del sistema, etc.)
Interesados internos (Arquitectos de Software, analistas de requerimientos)
Equipo (Líder, expertos en el dominio de la aplicación, expertos externos en arquitectura y secretario)

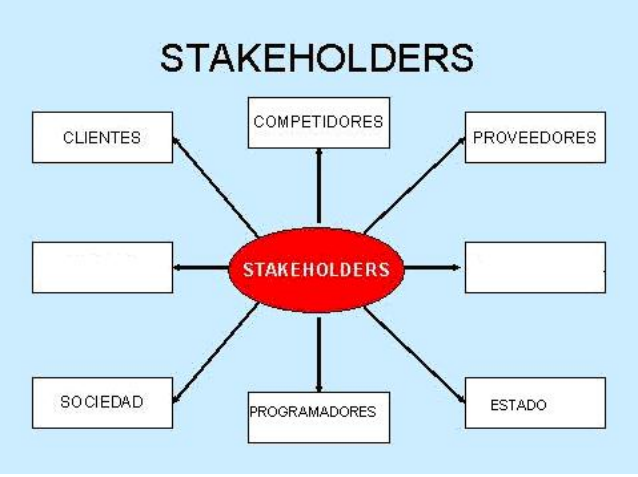

Fig.23. Stakeholders

Técnicas de evaluación

Existen varias técnicas para evaluar arquitecturas de software, que se clasifican en cualitativas y cuantitativas. Dentro de las técnicas de evaluación cualitativas se pueden utilizar: escenarios, cuestionarios o listas de verificación. Por otro lado, en las técnicas de evaluación cuantitativas se pueden emplear: métricas, simulaciones, prototipos, experimentos o modelos matemáticos.

La mayoría de los métodos de evaluación utilizan escenarios, que son secuencias específicas de pasos que involucran el uso o la modificación del sistema. Por lo regular, las técnicas de evaluación cualitativas son usadas cuando la arquitectura se encuentra en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada.

Planear o no la evaluación

Las evaluaciones pueden ser planeadas o no planeadas. Una evaluación planeada es aquella que ha sido contemplada dentro del ciclo de vida de desarrollo, por lo que es parte de las actividades del proyecto. Son varios los beneficios que se obtienen de realizar evaluaciones, algunos de estos son descubrir defectos e inconsistencias en la fase de diseño, asegurar que se cumplan los atributos de calidad, así como reducir los costos del proyecto. Comúnmente una evaluación no planeada, se presenta cuando la arquitectura de software contiene varios defectos que han sido detectados en etapas tardías del desarrollo, por ende, realizar una evaluación no planeada, representa retraso en los tiempos de entrega, así como un incremento en los costos del proyecto. Es recomendable que en el proyecto se contemple realizar una o varias evaluaciones a la arquitectura.

Resultado de la evaluación

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

¿Se ha diseñado la arquitectura más apropiada para el sistema?

¿Cuál de las arquitecturas propuestas es la más apropiada para el sistema a construir?

Además de responder estas preguntas, el reporte también indica el grado en que se cumplieron los atributos de calidad.

¿Qué resultado produce la evaluación de una Arquitectura?

La evaluación de una arquitectura no produce resultados cuantitativos. Además ayuda a encontrar debilidades.

¿Por qué cualidades puede ser evaluada una Arquitectura?

- Performance
- Availability
- Security
- Modifiability
- Reliability
- Portability
- Functionality
- Variability
- Subsetability
- Conceptual integrity

¿Cuáles son las salidas de una evaluación arquitectónica?

- Lista priorizada de los atributos de calidad requeridos para la arquitectura que está siendo evaluada.
- Riesgos y no riesgos.

¿Cuáles son los costos y beneficios de realizar una evaluación arquitectónica?

- Reúne a los stakeholders.
- Fuerza una articulación en las metas específicas de calidad.
- Fuerza una explicación clara de la arquitectura.

¿Cómo evaluamos una Arquitectura de Software?

Generalmente una AS se evalúa cuando ya está especificada, pero no se ha comenzado la implementación, pero una evaluación se puede hacer en cualquier momento, por ejemplo, existen las evaluaciones temprana y tardía. Dichas evaluaciones se hacen de manera cualitativa o cuantitativa.
Han sido utilizados diferentes enfoques para evaluar los requerimientos de calidad, por ejemplo;

- Evaluación basada en escenarios
- Simulación
- Modelado matemático
- Experiencia
- Métricas

Ejemplo de evaluación cuantitativa

La propiedad que ellos buscan evaluar es Modificabilidad (Mantenibilidad).

Ejemplo de evaluación cuantitativa

• Los sistemas fáciles de mantener son aquellos en donde el acoplamiento es bajo, la complejidad es baja y la cohesión entre componentes es alta.

La medida utilizada es acoplamiento entre módulos (CBM y CBMC)

¿Cómo medir complejidad?

• El método estudiado fue el de Zhao, el propone:

-Describir la AS a través de un lenguaje descriptivo de arquitectura (ADL), esto implica poder establecer una representación formal de la AS.

• Realizar un análisis de dependencias para arquitecturas.

-Dependencias compartidas
-Dependencias de flujo
-Dependencias de restricciones

• Crear una gráfica arquitectónica de dependencia, en estas gráficas hay vértices y arcos que unen los vértices (no da un ejemplo)
• Medir la complejidad a través de conteos sobre los arcos y vértices en las gráficas arquitectónicas de dependencia.

MÉTODOS DE EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE

De acuerdo con Kazman, hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos y no repetibles, lo que no brindaba mucha confianza. En virtud de esto, múltiples métodos de evaluación han sido propuestos. En este trabajo se explican algunos de los más importantes. Ver anexo #2.

Comparación entre métodos de evaluación

La tabla 7 presenta una comparación entre los métodos de evaluación Software Architecture Analysis Method (SAAM), Architecture Trade-off Analysis Method (ATAM), Active Reviews for Intermediate Designs (ARID), Modelo de Negociación WinWin, Cost-Benefit Analysis Method (CBAM).

	ATAM	SAAM	ARID	WinWin	CBAM
Atributos de Calidad contemplados	-Modificabilidad -Seguridad -Confiabilidad -Desempeño	-Modificabilidad -Funcionalidad	-Conveniencia del diseño evaluado	-Funcionalidad	-Funcionalidad -Modificabilidad

Objetos analizados	-Estilos arquitectónicos -Documentación -Flujo de datos -Vistas Arquitectónicas	-Documentación -Vistas Arquitectónicas	-Especificación de los componentes	-Conflictos entre requerimientos	-Estilos Arquitectónicos -Documentación
Etapas del proyecto en las que se aplica	-Luego de que el diseño de la arquitectura ha sido establecido	-Luego de que la arquitectura cuenta con funcionalidad ubicada en módulos	-A lo largo del diseño de la arquitectura	-Luego de que el diseño de la arquitectura ha sido establecido	-Luego de que el diseño de la arquitectura ha sido establecido
Enfoques utilizados	-Utility Tree y lluvia de ideas para articular los requerimientos de calidad -Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos	-Lluvia de ideas para escenarios y articular los requerimientos de calidad -Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios	-Revisiones de diseños, lluvia de ideas para obtener escenarios	-Teoría “W”	-Teoría “W” -Análisis de escenarios

Tabla 7. Comparación entre métodos de evaluación.

Evaluando la arquitectura propuesta

Los atributos de calidad de un software están fuertemente influenciados por la arquitectura del sistema. Muchos de estos atributos tales como mantenibilidad, desempeño, usabilidad, modularidad, etc., no pueden ser medidos directamente. Sin embargo, la experiencia señala que la arquitectura de software propicia algunos de ellos. Por lo tanto, la arquitectura del software es clave para la calidad. Por esto, un análisis de la arquitectura debe ser ejecutado para determinar cuán satisfactoria es para el propósito del sistema. Para realizar esta estimación de la calidad a partir de la arquitectura, algunos métodos, tales como ATAM, ABDM, ARID, etc., los cuales incorporan diferentes técnicas de evaluación, son utilizados. Estas técnicas incluyen los escenarios, la simulación, los modelos matemáticos, el prototipo, entre otros, los cuales permiten una evaluación temprana. Sin embargo, no todas las técnicas son aplicables y efectivas para cualquier atributo de calidad. En el presente trabajo se hace una evaluación para determinar el cumplimiento de los requerimientos de calidad que engloban la "utilidad" del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios.

La arquitectura que se propone en este trabajo es representada a un nivel medio (diseño bastante detallado en algunas vistas y poco detallado en otras), ante esta situación, y la necesidad de evaluación en las fases tempranas del diseño, se propone la utilización de las características que proveen tanto ADR como ATAM por separado. De ADR, resulta conveniente la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID para efecto de la evaluación temprana de los diseños de una arquitectura de software. Este es el método utilizado en la evaluación de este trabajo. Debido a todo esto, se decidió validar la arquitectura evaluando como las vistas seleccionadas responden a cada uno de los principales requerimientos y restricciones del sistema.

Constituyendo nuestros escenarios los siguientes:

- Rendimiento del sistema.
- Modificabilidad y evolución del sistema.
- Seguridad del sistema.
- Disponibilidad del sistema.
- Portabilidad del sistema.

- Independencia entre la aplicación y los sistemas gestores de bases de datos.

Rendimiento del sistema: Se debe consultar la vista de procesos para analizar la concurrencia en el sistema. También se debe consultar la vista de despliegue.

Modificabilidad y evolución del sistema: Para analizar el impacto de un cambio la vista lógica es muy útil, donde nos ayuda mucho la descripción de la arquitectura vertical de alto nivel y la vista refinada de la arquitectura del sistema, que muestran como un cambio en una capa más baja puede ser ocultado detrás de sus interfaces y no impactará las capas encima de ella, también nos es muy útil la descripción de la arquitectura horizontal y la vista de implementación.

Seguridad del sistema: La vista de despliegue es usada para ver los puntos de contacto del sistema con el exterior, así como el software encargado de establecer la interacción; también muestra donde los aspectos autenticación e integridad son manejados, el servidor de aplicaciones Web de intranet realiza esta tarea.

Disponibilidad del sistema: La vista de procesos puede ayudar a analizar los procesos de mayor concurrencia. La vista de despliegue es usada para mostrar posibles puntos de fallo. Pueden definirse números de fiabilidad para un módulo como una propiedad en la vista refinada de la arquitectura del sistema.

Portabilidad del sistema: El sistema será desarrollado utilizando la plataforma de desarrollo PHP, la cual es compatible con los servidores como apache e IIS, funciona en plataformas diferentes como el Linux, el Windows y el Unix, soporta gran variedad de bases de datos como MySQL, el Informix, el Oracle, el Sybase, Sólido, PosgreSQL y ODBC Genérico.

Una vez culminada la evaluación, se puede concluir acerca de la efectividad de la propuesta de arquitectura utilizada, encontrando que, es apropiada para aquellos atributos de calidad observables en tiempo de ejecución.

Conclusiones del capitulo

Después del análisis de este capítulo se puede ver que en la AS se sigue teniendo el gran problema de que la recolección de datos no es difícil, pero si la selección de la métrica adecuada y la interpretación de los datos recolectados. También se puede ver que la posible razón de que el progreso sea lento en el desarrollo y evolución de los sistemas de software, es que empleamos carpinteros y constructores, en vez de arquitectos, no se utilizan las personas que verdaderamente están preparadas para desempeñar este trabajo.

En la AS una evaluación temprana puede ser aplicada a sólo una AS o a un grupo de AS que compiten entre si. En el último caso, la evaluación revela las debilidades y fortalezas de cada una.

Problemáticas a las que se enfrenta la AS:

- Existe poco material.
- Difícil comprensión de los artículos.
- No hay ejemplos prácticos.

Por todo lo antes visto se puede decir que por la experiencia personal se recomienda que:

- Hay que reforzar conocimientos.
- Hay que acoplarse a trabajar en equipo.
- Adquisición de mayores habilidades para el manejo de conceptos de Ingeniería de Software.
- Agrupar un grupo mayor de 2 arquitectos para el desarrollo de esta.

Conclusiones

En este trabajo se demuestra que la Arquitectura Orientada a Servicios (SOA) es el estilo existente que da mejor solución a la hora de implementar Servicios Web. Este estilo permite el crecimiento tecnológico y organizacional de forma óptima para las aplicaciones. Con este estilo se separa fácilmente la capa de datos de la presentación de la lógica de las aplicaciones.

La propuesta realizada ha permitido establecer de forma correcta la arquitectura del sistema que se desea desarrollar, todo lo expuesto en dicha propuesta es comprendido por los demás desarrolladores y sirve como base para la construcción de la aplicación. Las vistas seleccionadas para desarrollar la arquitectura se hicieron teniendo en cuenta las necesidades de los implicados en el desarrollo de la aplicación, con dichas vistas se le da solución a las necesidades primarias de información de los desarrolladores. La evaluación de la arquitectura demuestra que se ha realizado una arquitectura que responde a los requisitos de calidad propuestos, pues con esta arquitectura se da solución a un diseño que permitirá desarrollar la aplicación para automatizar la Casona 23yB.

Recomendaciones

Con el objetivo de mejorar y ampliar en la documentación que se presenta en este trabajo se recomienda.

Incluir en un futuro los diagramas de clases y de componentes de los demás módulos que faltan, pues no fueron incluidos en este trabajo por no ser objetivo.

Profundizar más en la vista de procesos para que se tenga una mejor comprensión de los mismos, y así entender más el funcionamiento de los Servicios Web.

Estudiar profundamente otros CMS principalmente en Java, para que en un futuro se pueda mejorar la forma de consumir los Servicios Web y así lograr una mejor integración con estos.

Para establecer una documentación estándar a la hora de implementar una Arquitectura Orientada a Servicios, se recomienda que se estudie con más profundidad la propuesta de la Dra. Sofía Álvarez Cárdenas en su trabajo “Diseño de la arquitectura y los Servicios Web”.

Aumentar el número de arquitectos a la hora de desarrollar otro proyecto, logrando así una mejor división del trabajo, la cual ayude a realizar un trabajo más rápido, profundo y abarcador sobre la arquitectura de un sistema.

Bibliografía

[BCK98] Len Bass, Paul Clements y Rick Kazman. Software Architecture in Practice. Reading, Addison-Wesley, 1998.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal. Pattern-oriented software architecture – A system of patterns. John Wiley & Sons, 1996.

[BR01] Jason Baragry y Karl Reed. “Why We Need a Different View of Software Architecture”. The Working IEEE/IFIP Conference on Software Architecture (WICSA), Amsterdam, 2001.

[Bro75] Frederick Brooks Jr. The mythical man-month. Reading, Addison-Wesley, 1975.

[Cle96a] Paul Clements. “A Survey of Architecture Description Languages”. Proceedings of the International Workshop on Software Specification and Design, Alemania, 1996.

[Cle96b] Paul Clements. “Coming attractions in Software Architecture”. Technical Report, CMU/SEI-96-TR-008, ESC-TR-96-008, Enero de 1996.

[Dij68a] Edsger Dijkstra. “The Structure of the THE Multiprogramming system.” Communications of the ACM, 26(1), pp. 49-52, Enero de 1983.

[DK76] Frank DeRemer y Hans Kron, “Programming-in-the-large versus programming-in-the-small”. IEEE Transaction in Software Engineering, 1976.

[Fie00] Roy Thomas Fielding. “Architectural styles and the design of network-based software architectures”. Tesis doctoral, University of California, Irvine, 2000.

[GoF95] Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Design Patterns: Elements of reusable object-oriented software. Reading, Addison-Wesley, 1995.

[GS94] David Garlan y Mary Shaw. “An introduction to software architecture”. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.

[NATO76] I. P. Sharp. Comentario en discusión sobre teoría y práctica de la ingeniería de software en conference de NATO Science Committee, Roma, 27 al 31 de Octubre de 1969. Software Engineering Concepts and Techniques: Proceedings of the NATO conferences. J. N. Bruxton y B. Randall (eds.), Petrochelli/Charter, 1976.

[Par72] David Parnas. “On the Criteria for Decomposing Systems into Modules.” Communications of the ACM 15(12), pp. 1053-1058, Diciembre de 1972.

[Pfl02] Shari Lawrence Pfleeger. Ingeniería de Software: Teoría y Práctica. Madrid, Prentice-Hall.

[PW92] Dewayne E. Perry y Alexander L. Wolf. “Foundations for the study of software architecture”. ACM SIGSOFT Software Engineering Notes, Octubre de 1992.

[SG96] Mary Shaw y David Garlan. Software Architecture: Perspectives on an emerging discipline. Upper Saddle River, Prentice Hall, 1996.

[Spo71] C. R. Spooner. “A Software Architecture for the 70's: Part I - The General Approach.” Software - Practice and Experience, 1 (Enero-Marzo), 1971.

[Wir71] Niklaus Wirth. “Program development by stepwise refinement”, Communications of the ACM, Abril de 1971.

Web Services ORG, http://www.Webservices.org (http://es.wikipedia.org/wiki/Servicio_Web)

World Wide Web Consortium (W3C), http://www.w3.org/
(http://es.wikipedia.org/wiki/World_Wide_Web_Consortium)

URI. World Wide Web Consortium (W3C), “Uniform Resource Identifier (URI)”, www.w3.org/Addressing/

XML. World Wide Web Consortium (W3C), “eXtensible Markup Language (XML)”, http://www.w3.org/XML

SOA. “Service Oriented Architecture”, http://www.service-architecture.com/

World Wide Web Consortium (W3C), “Simple Object Access Protocol (SOAP)”: http://www.w3.org/TR/SOAP/ y http://www.develop.com/soap

World Wide Web Consortium (W3C), “Web Services Description Language (WSDL)”: http://www.w3.org/TR/wsdl.

Business Process Execution Language (BPEL), http://www.service-architecture.com/

Java Remote Method Invocation (Java RMI), www.java.sun.com/products/jdk/rmi/

Common Object-Request Broker Arquitectura (CORBA), www.corba.org/

Universal Description, Discovery and Integration (UDDI): http://www.uddi.org , http://uddi.microsoft.com/ y http://www-3.ibm.com/services/uddi/

http://www.enterate.unam.mx/Articulos/2006/febrero/arquitec.htm

http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp#1

http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_Web

http://es.wikipedia.org/wiki/Programaci%C3%B3n_Extrema

http://deigote.blogspot.com/2006/03/extreme-programming.html

http://es.wikipedia.org/wiki/Rational_Unified_Process

http://www.reynox.com.ar/sap/metodologia.php

http://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_base_de_datos

http://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_contenido

http://geneura.ugr.es/~jmerelo/tutoriales/cms/

http://www.objectis.org/about-es/plone

http://javalibre.com/

http://lfc.uah.es/DAW/

http://216.239.51.104/search?q=cache:ewP55KPWb6EJ:www.lifia.info.unlp.edu.ar/papers/2001/Silva2001.pdf+importancia+%2B+%22Aplicaciones+Web%22&hl=es&ct=clnk&cd=2&gl=cu

http://www.maestrosdelWeb.com/editorial/ajax/

http://blogs.clearscreen.com/nandy/archive/2004/06/15/335.aspx

http://www.therationaledge.com/content/jan_01/f_rup_pk.html

http://216.239.51.104/search?q=cache:0DxBT-KLFm0J:www.dcc.uchile.cl/~luguerre/cc61j/recursos/clase2.ppt+RUP&hl=es&ct=clnk&cd=5&gl=cu

http://www.desarrolloWeb.com/faq/504.php

http://www.agapea.com/SISTEMAS-GESTORES-DE-BASES-DE-DATOS-n23806i.htm

http://www.error500.net/garbagecollector/archives/categorias/bases_de_datos/sistema_gestor_de_base_de_datos_sgbd.php

http://mosaic.uoc.edu/articulos/cms1204.html

http://cms-hispano.org/index.php?s=content&p=cms

http://diario.grumpywolf.net/2003/11/pero-que-es-un-cms/

Maria Cecilia Bastarrica, David Gomez, Cristian Wilckens. "Autómatas como Lenguaje de Definición de Arquitecturas". Fac. Ing- Univ Tarapacá. Octubre 2004. Num Pag 11

Dra. Sofía Álvarez Cárdenas. "Diseño de la Arquitectura y los Servicios Web". Instituto Superior Politécnico "José Antonio Echeverría"

Daniel Perovich, Andrés Vignaga. "SAD del Subsistema de Reservas de Gestión Hotelera". Grupo COAL Instituti de Computación Fac de Ing. Universidad de la Republica. Montevideo Uruguay.

Grupo de Investigación en Ingenieria de Sofware. "La Importancia de la Arquitectura en el Desarrollo de Software de Calidad". Universidad EAFIT. Febrero 17. 2005

Hernán Astudillo. "Five Ontological Levels To Describe and Evaluate Software Architectures". Fac. Ing- Univ Tarapacá. Noviembre 2004

Aquilino A. Juan Fuente. “Arquitectura de Software Métodos Prácticos”. Departamento de Informática Lenguajes y Sistemas Informáticos Universidad de Oviedo

Rosa Virginia Icedo Ojeda, Jorge Moisés Trejo Vargas. “Software Arquitectura Assesment”. CIMAT. Mayo 2003

Peña, Q.; Talavera, M.; Grimán, A.; Pérez, M. y Mendoza, L. “Evaluación Arquitectónica de Calidad de Software Usando la Técnica de Prototipo”. Dpto de Procesos y Sistemas. Universidad Simón Bolívar.2002

Aleksander González, Marizé Mijares, Luis E. Mendoza, Anna Grimán, María Pérez. “Método de Evaluacion de Arquitecturas de Software Basada en Componentes”. Universidad Simón Bolívar. 2005

Ramon Molineda. “Arquitectura de software: arte y Oficio”. Febrero 2005. Disponible en: http://Web.iti.upv.es/actualidadtic/2005/02/2005-02-arquitectura.pdf

Esteban Malsano. “Descripción de la arquitectura para Software Browser Link-all”. Disponible en: http://www.fing.edu.uy/inco/proyectos/linkall/ingsoft/doc/Arquitectura%20Browser.pdf

Sebastián Javier Marconi. “Persistencia de un Modelo de Objetos en una Base de Datos Relacional”. Octubre 2004. Disponible en: http://k7k0.apihuella.com/tesis/mapeo_objeto_relacional.pdf

Erika Cmacho, Fabio Cardeso, Gabriel Nuñes. Arquitecturas de Software, Guía de Estudio. Abril 2004

Francesc Daniel Muñoz. HIDRA: Invocaciones Fiables y Control de Concurrencia. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. 2004

Billy Reinoso. “Architect Academy: Seminario de Arquitectura de Software”. Universidad de Buenos Aires

Glosario de términos

API: Application Programming Interface (Interfaz de Aplicación para Programación).

Capas: Es uno de los patrones de diseño más utilizado para cualquier tipo aplicaciones, en este, básicamente se divide los elementos de diseño en paquetes de Interfaz de Usuario, Lógica de Negocio y Acceso a Datos y Servicios. Se describen las diversas capas del sistema y su contenido, detallan los límites entre las capas y las reglas preestablecidas para cada una.

Canal Educativo: Se crea para el proyecto Universidad para todos, el 9 de mayo del 2002, para apoyar la docencia, el sistema educativo nacional y el desarrollo de la cultura. En los Canales Educativos la dirección de la Revolución Cubana tiene puesto todo el empeño y el interés en que va a ser un aporte de enorme utilidad social, intelectual y educativo. Ahora se puede demostrar el potencial pedagógico de la educación cubana, y la fuerza popular de grandes masas interesadas en el aprendizaje.

Deployment: Despliegue.

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos).

ISO: International Organization for Standardization (Organización Internacional para la Estandarización).

NTIC: Nuevas Tecnologías de la Información y de las Comunicaciones.

Universidad para todos: Un proyecto de enseñanza general de cultura masiva; trascendente en el ámbito de la instrucción universal y la batalla de ideas. Se trata del aula más inmensa de la cual se tenga noticia en el mundo

Vista Lógica: Muestra los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada en la plataforma de ejecución.

Describe la solución en términos de paquetes y clases de diseño. Dentro de esta vista se describen los casos de uso, subsistemas, paquetes y clases de los casos de uso más significativos arquitectónicamente, así como también se señalan las funciones, relaciones, atributos y operaciones de estos.

Vista de Implementación: La vista de implementación muestra en general las dependencias y cómo se implementan los componentes físicos del sistema, agrupándolos en subsistemas organizados en capas y jerarquías.

Vista de casos de uso: Describe el proceso de negocio más significativo y el modelo del dominio. Presenta los actores y los casos de uso para el sistema.

Vista de Procesos: describe los procesos concurrentes del sistema.

Vista de Deployment: Presenta aspectos físicos como topología infraestructura informática, e instalación de ejecutables. Incluye además plataformas y software de base.

WSDL: Web Services Description Language (Lenguaje de descripción de los Servicios Web.)