Universidad de las Ciencias Informáticas

Facultad 6

Título: Arquitectura de Software para el desarrollo de productos del Departamento de Integración de Soluciones según el modelo de negocio *Software como Servicio.*

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Jorge Ernesto Fernández de la Torre.

Tutores: MBA. Yosdenis Urrutia Badillo.
Ing. Cecilia Milián Delgado.

Consultantes: Ing. Armando Robert Lobo.
Ing. José Ángel Martínez García.

La Habana, junio de 2012.
"Año 54 de la Revolución"

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 _____.

Jorge Ernesto Fernández de la Torre

_____

Autor

MBA. Yosdenis Urrutia Badillo

_____

Tutor

Ing. Cecilia Milián Delgado

_____

Tutora

Datos de contacto.

Autor: Jorge Ernesto Fernández de la Torre.
Universidad de las Ciencias Informáticas, La Habana, Cuba.
E-mail: jefernandez@estudiantes.uci.cu

Tutor: MBA. Yosdenis Urrutia Badillo.
Universidad de las Ciencias Informáticas, La Habana, Cuba.
E-mail: yosdenis@uci.cu

Tutora: Ing. Cecilia Milián Delgado.
Universidad de las Ciencias Informáticas, La Habana, Cuba.
E-mail: cmilian@uci.cu

Resumen.

El mundo del software se presenta como un entorno altamente evolutivo. El surgimiento de nuevas tecnologías favorece la creación de aplicaciones cada vez más robustas y eficientes, elevando el nivel competitivo empresarial en el sector. Conocer las tendencias de mayor impacto en el mercado permite adoptar las estrategias de desarrollo correctas, basadas en modelos de negocio que maximicen las ganancias de la organización. En este sentido, el Software como Servicio constituye una atractiva propuesta para el comercio de sistemas web, relativamente joven, pero lo suficientemente probada para determinar su solidez. Con este trabajo se presenta una arquitectura de software genérica, basada en los principios aplicados en el modelo mencionado, a fin de ampliar el horizonte comercial de los productos desarrollados por el Departamento de Integración de Soluciones. Se describen en ella las tecnologías más representativas (en su mayoría libres) para el desarrollo de aplicaciones web enriquecidas, entre las cuales se encuentran Linux, PostgreSQL, Apache, PHP y AJAX. Se plantean también los estilos y patrones arquitectónicos a emplear para obtener de conjunto una arquitectura robusta, escalable, extensible y multiplataforma que caracterice los productos desarrollados por el departamento. La arquitectura candidata propuesta ha sido evaluada mediante el método de pruebas para arquitecturas ATAM, demostrando que cumple cabalmente con los objetivos para los cuales fue concebida.

Palabras claves: Arquitectura de Software, Arquitectura Orientada a Servicios, modelos de negocio, Software como Servicio.

Índice de contenidos.

Índice de Tablas.

Índice de Figuras.

Introducción.

A partir del auge experimentado por las Tecnologías de la Información y las Comunicaciones (Tics), la sociedad ha llegado a un punto en el que no se concibe la realización de gran parte de nuestro quehacer diario sin el empleo de algún medio tecnológico que contribuya a la automatización de las tareas en cuestión. Este fenómeno se puede observar en ramas como la economía, la salud, la educación, entre otras, donde las herramientas informáticas han tomado un papel decisivo en la agilización y mejora de la calidad de todos los procesos sociales y productivos.

Con la aparición de la Internet, la forma en que las relaciones económicas se llevaban a cabo tradicionalmente experimentó un salto hacia adelante. Las mesas de negociaciones en muchas ocasiones se vieron sustituidas por páginas web donde clientes y productores exponen sus ofertas y demandas sin la existencia de mediadores, fronteras de idioma o distancia que los separen. El Comercio Electrónico o E-Bussines como se le conoce mundialmente ha cambiado por completo el modo de actuar de las empresas. Hoy día no se concibe una empresa exitosa que no tenga un sitio web donde promocionar sus productos, además de toda la infraestructura que viene acarreada al proceso de soporte, aseguramiento, seguimiento, transportación y comercialización de sus mercancías.

Cuba independientemente de su condición de nación bloqueada económicamente por la mayor potencia mundial de nuestra época, hace todo lo posible por no quedar rezagado en cuanto a las últimas tendencias tecnológicas. El desarrollo de las mismas constituye un factor fundamental para ubicar el fruto del trabajo del capital humano en un lugar de prestigio ante los ojos del mundo. Producto de este pensamiento, materializado por el Comandante en Jefe, se crea la Universidad de las Ciencias Informáticas (UCI), centro de vanguardia en Cuba que vincula la formación de profesionales revolucionarios con la producción de software y servicios. La creación de esta universidad tiene como premisa que al desarrollar la Informática, esta se convierta en una importante fuente de ingreso de divisas al país, a la vez que se lleven a cabo los procesos de automatización que requiere la sociedad cubana.

Insertado en la propia institución se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC), al cual pertenece el Departamento de Integración de Soluciones, responsable de importantes proyectos de informatización destinados tanto al país como a naciones extranjeras. Asociado al mismo se encuentra el Grupo de Mercadotecnia, encargado de llevar a cabo los estudios que permitan una buena inserción de sus productos al mercado, garantizado así la mayor obtención de ganancias posible. En el departamento actualmente sólo se desarrollan productos hechos a la medida cuando algún cliente viene en busca de una determinada herramienta de automatización. Esta

situación constituye un freno para su potencial productivo, que se ve limitado por la existencia de una estrategia que contempla una única línea de desarrollo/comercialización, sin tener en cuenta otras tácticas novedosas aplicadas hoy día en el mercado internacional. Todo esto trae como consecuencia que en muchas ocasiones se desaprovechen oportunidades que pueden aportar resultados positivos con su explotación.

Por todo lo antes expuesto, se establece como **problema de la investigación** la siguiente interrogante: ¿Cómo mejorar el proceso de comercialización de los productos desarrollados por el Departamento de Integración de Soluciones?

El **objeto de estudio** de la investigación está conformado por: la Arquitectura de Software aplicada al modelo negocio Software como Servicio (SaaS), teniendo como **campo de acción** las vistas de la arquitectura sobre las cuales incide SaaS como modelo de negocio.

Se plantea como **objetivo general:** diseñar las vistas de la arquitectura que permitan el desarrollo de los productos creados por el Departamento de Integración de Soluciones según el Modelo de Negocio SaaS.

Se definen como **objetivos específicos** para llevar a cabo esta investigación:

-Realizar un estudio de las tecnologías existentes que respondan al modelo de negocio SaaS.
- Diseñar las vistas de la arquitectura que se ajusten a las características del modelo de negocio.
- Validar la concepción de las vistas de la arquitectura propuestas.

Para dar cumplimiento a cada uno de los objetivos se plantean como **tareas de la investigación:**

- Estudio del Estado del Arte de los modelos de negocio en el proceso de comercialización del software a nivel internacional y nacional.
- Valoración de los principales competidores de los productos a comercializar por el Departamento de Integración de Soluciones.
- Selección de las herramientas que se ajusten a los productos desarrollados por el Departamento de Integración de Soluciones, según el modelo SaaS.
- Identificación de los elementos arquitectónicos que incidan sobre el modelo de negocio.
- Descripción de las vistas de la arquitectura relevantes para el desarrollo de aplicaciones informáticas según el modelo de negocio SaaS.
- Validar la arquitectura genérica propuesta.

Como **posible resultado** se esperan obtener las vistas de la arquitectura que den soporte a la futura implementación del Modelo de Negocio SaaS sobre los productos del Departamento de Integración de Soluciones.

Atendiendo a estos criterios, el documento para esta investigación queda estructurado de la siguiente forma:

Capítulo 1: Fundamentación Teórica.
En este capítulo se da a conocer una explicación acerca del estado del arte de la investigación y las tendencias a nivel nacional e internacional en cuanto a las distintas tecnologías y procedimientos de mayor utilización en el desarrollo de software de tipo SaaS, haciendo énfasis en sus características. También se abordan temas relacionados con la comercialización del software a nivel nacional e internacional y se hace un breve estudio de los principales competidores de los productos a comercializar por el Departamento de Integración de Soluciones.

Capítulo 2: Definición de las vistas para la arquitectura.
Este capítulo recoge los elementos arquitectónicos significativos que corresponden al modelo de negocio SaaS. A partir de estos se hará el diseño de las vistas de mayor repercusión sobre la arquitectura para el desarrollo de las aplicaciones del Departamento de Integración de Soluciones según el modelo de negocio seleccionado.

Capítulo 3: Evaluación de la arquitectura propuesta.
Una vez culminadas las vistas que conforman la arquitectura genérica propuesta, se realizan las pruebas a la arquitectura a través del método ATAM (basado en escenarios).

Capítulo 1: Fundamentación Teórica.

Este capítulo tiene como objetivo la realización de un estudio referente al estado del arte de los modelos de negocio, haciendo énfasis en el Software como Servicio. También se hará una presentación de los elementos pertenecientes a la Arquitectura de Software (dígase estilos y patrones arquitectónicos, tecnologías, herramientas de apoyo) que se encuentran estrechamente relacionados a la implementación del modelo SaaS en cuestión.

1.1 Modelos de negocio.

Se define un modelo de negocio (MN) como la forma en que las empresas obran con el fin de obtener ganancias y beneficios. Por tanto, encierran la estrategia para desarrollar actividades vitales como la búsqueda de nuevos clientes, definir productos de buena aceptación en el mercado y las vías a emplear para promocionar los mismos (estrategias de marketing, publicidad, etc.), entre otros elementos organizacionales a tener en cuenta.

Existen ejemplos tradicionales de MN tales como el del **modelo del tendero**, el cual consiste en ubicar la empresa o establecimientos en lugares de mucha afluencia de público, a fin de desplegar sus ofertas con una mayor oportunidad de comercialización. Otro modelo clásico se tiene en el modelo del **cebo y el anzuelo**, utilizado por muchas compañías a lo largo de la historia, basado en el principio de la venta de un producto relativamente barato, que utilice recambios o piezas que vender luego a precios muy altos (ej. la impresora y los tóneres).

Con el tiempo, estas formas de hacer comercio se han ido perfeccionando y adaptando a los diferentes entornos propuestos por el mercado. Un ejemplo actual del que se han beneficiado numerosas empresas llevándolo a la práctica es el modelo **Canvas**, planteado por el Dr. Alex Osterwalder (1). El mencionado autor, partiendo de tres preguntas básicas (¿Cómo?, ¿Qué?, y ¿Quién?), confeccionó un esquema donde define los elementos fundamentales para establecer una plataforma de negocios sólida, aplicable a cualquier entorno empresarial (ver Anexos, Figura 12).

1.1.1 El modelo de negocio SaaS.

Según un conjunto de definiciones de varios autores, SaaS (Software as a Service o Software como Servicio) es un modelo de distribución del software a través de las redes o Internet, por lo cual no influyen la ubicación geográfica del cliente y la empresa. Brinda al cliente la oportunidad de contar con servicios de aplicaciones libres de costos de mantenimiento, operaciones técnicas y soporte, optimizando sus gastos y gestión de recursos.

Surge bajo una idea similar al alquiler de bienes o servicios, por ejemplo, Ud. viaja por una semana a algún sitio distante, donde pretende permanecer por un espacio de tiempo, y el costo de la renta de un auto evidentemente es mucho más barato que adquirir uno. Bajo esta concepción, además, las empresas pueden obtener mayores ganancias (o una entrada más estable de ingresos) a partir del alquiler del software por un precio más módico en lugar de percibir frutos cuando algún cliente decide comprar el software.

SaaS tiene como predecesor al MN ASP (Aplication Service Providers o Proveedor de Servicios y Aplicaciones), con el cual comparte desde la perspectiva del cliente, características similares, llegando incluso a confundir a expertos en el tema afirmando que son lo mismo, algo que no es correcto. Ambos modelos aportan al cliente ventajas similares, ya que los costos y puesta en marcha de los sistemas es menor, al igual que el costo por concepto de desarrollo de aplicaciones. Al emplear soluciones de este tipo se obtienen aplicaciones ya desarrolladas con las condiciones o requerimientos que necesita el cliente para sus gestiones, a la par que evita los riesgos que conlleva el hacer sus propias aplicaciones "en casa". Otra ventaja radica en que no es necesaria una gran inversión en tecnología de servidores, dígase hardware, mantenimiento y personal especializado que interactúe con los mismos, con lo cual se logra una "externalización de costes", al poner en manos de terceros la realización de estas actividades. Estos elementos citados protegen a la empresa además, del riesgo que suponen los cambios tecnológicos, al no tener que invertir en hardware y software que en un momento puedan quedar obsoletos, siendo responsabilidad de la empresa proveedora la actualización de estos bienes. Teniendo cubiertas sus espaldas con todas estas facilidades, el cliente puede dedicarse entonces a su quehacer real, librándose de aspectos técnicos que no constituyen el centro de sus negocios.

Ahora bien, una vez identificadas las similitudes existentes entre SaaS y ASP, vamos a los aspectos que hacen que difieran uno del otro. En primer lugar, con SaaS desaparecen las licencias, siendo el pago por consumo o uso el modo de facturación empleado. Existe el hecho de que muchos ASP necesitan del despliegue de módulos adicionales del lado del cliente, cosa que no sucede en SaaS donde todas las aplicaciones se encuentran en los servidores del proveedor. Quizás el aspecto más significativo es la tecnología introducida por SaaS: "Multitenancy", que no es más que la capacidad de dar servicio con una misma infraestructura (servidor de BD y aplicaciones) a varios suscriptores, a diferencia de ASP que utiliza generalmente uno para cada cliente, lo cual permite lograr (con SaaS) una escalabilidad sin límites. Esto trae consigo una serie de factores que acentúan aún más las diferencias entre uno y otro, por ejemplo: es mucho más fácil actualizar y dar mantenimiento a una infraestructura que da servicio a muchos clientes (SaaS), que hacerlo para cada conjunto de servidores que se tenga por cliente (ocurre muchas veces en ASP), por lo que el período con que se realizan modificaciones en las aplicaciones en favor de los afiliados equidista mucho de uno modelo a

otro. A la vez, es más probable que se produzcan errores durante el proceso en ASP, por el simple hecho de que se han de realizar operaciones en cada uno de los servidores que se tienen destinados para cada cliente, incrementando la posibilidad de que ocurra algún fallo. Como es lógico también, a una empresa que se base en ASP, le resulta más costoso mantener al día su infraestructura tecnológica y darle mantenimiento, tema que luego pasa cuenta al cliente en la factura, ya que a mayor costo de producción de un servicio, más elevado resulta el precio que se cobra por el mismo.

A partir de esta comparación pueden apreciarse las principales ventajas que trae el uso de SaaS para las empresas y sus clientes, sin embargo han de tenerse presente algunas dificultades. Para poder ofrecer un software como servicio, es necesaria una conexión estable, ya que no se concibe SaaS sin la mediación de la red o Internet como canal de comunicación entre el cliente y las aplicaciones del proveedor, que se encuentran alojadas en los servidores de la empresa de servicios. El índice de personalización de las aplicaciones para con los clientes es menor, ya que están orientadas a un uso por parte de muchos clientes, aunque este factor no impide que un buen uso de interfaces con una estructura bien pensada y estandarizada haga más ameno el trabajo de los consultantes. Y por supuesto, está el tema más apremiante de todos, referente a la seguridad de la información, dada la responsabilidad que asume el proveedor del servicio al tener en sus manos información sensible de sus clientes, la cual debe proteger celosamente y garantizar su integridad y acceso las 24 horas cada vez que un usuario lo requiera. Además, la empresa debe estar preparada para enfrentar con el tiempo un incremento en la escalabilidad, en la medida en que se le vayan sumando nuevos afiliados que requieran de su sistema un mayor nivel de cómputo y almacenamiento, sin que se vea afectado el rendimiento de las aplicaciones.

Si bien a la hora de desarrollar aplicaciones de tipo SaaS en nuestro país no existe la posibilidad de dar servicios a terceras naciones por las condiciones técnicas en las que se encuentra inmersa la isla, existen un buen número de empresas locales que pueden ver beneficiadas sus actividades (experimentando un aumento en organización y productividad) a través de contratos con la UCI. Dadas las circunstancias actuales, está de más decir que en estos momentos si la UCI no es el único centro que se encuentra en condiciones de mantener servidores de aplicaciones con los cuales dar servicios a entidades externas, sobrepasa a sus posibles competidores dentro del país en cuanto a parque tecnológico, infraestructura de redes y servidores se trata. De materializarse este hecho, puede significar un nuevo flujo de ingresos a la universidad, haciéndose extensivo a la economía nacional. Esto permitiría además automatizar los procesos de una gran cantidad de empresas cubanas, sin grandes conflictos en cuanto a despliegue de aplicaciones y servicios de asistencia postventa, ya que muchas de estas labores pueden manejarse de forma centralizada desde la propia universidad. Para el país esto representaría un considerable ahorro de divisas al no tener que llevarse a cabo un mayor

cúmulo de inversiones en hardware en cada organización; aspecto que va acorde a la nueva reestructuración de la economía que se viene realizando a partir de los lineamientos aprobados por el Partido Comunista de Cuba y que puede lograrse de esta manera.

1.2 Panorámica de mercado de los productos del Departamento de Integración de Soluciones.

Este epígrafe está destinado a la realización de una breve caracterización de mercado para los principales productos desarrollados por el Departamento de Integración de Soluciones, específicamente el Sistema Integrado de Gestión Estadística (SIGE) y el Generador Dinámico de Reportes (GDR).

Como reseña particular de ambos sistemas, puede decirse que han tenido como bases de su surgimiento acuerdos tomados entre la Oficina Nacional de Estadísticas (ONE) y la Universidad de las Ciencias Informáticas (UCI). El fin de estos tratos deviene en lograr un mayor índice de rendimiento y renovación a la vez, de las herramientas con que cuenta la primera para el desarrollo normal de sus actividades.

El primer sistema citado (SIGE), es una aplicación destinada a la captura, digitalización y validación de datos estadísticos para su posterior explotación en la toma de decisiones empresariales. Su funcionamiento parte del análisis de datos obtenidos en formularios matriciales, que contienen indicadores de los aspectos de mayor impacto en el ámbito empresarial y que permiten evaluar el progreso que se tiene en cada una de las metas fijadas. Está integrado por los Módulos de Registros y Clasificadores, Generador de Modelos y Encuestas, Entrada de Datos y Reportes. De estudios realizados con anterioridad, se ha determinado que otros sistemas de características similares, presentan inconvenientes que no los hacen factibles para su uso en el país, dado su carácter privativo y el alto costo de sus licencias, o bien por no ser orientados a funcionar en múltiples plataformas (CSPro, SIDEMO, DV\SURVEY).

El GDR por su parte, es una herramienta desarrollada con tecnologías web (Symfony y PHP), multiplataforma, que permite la generación de reportes a partir de consultas realizadas a múltiples gestores de bases de datos. Los informes que se generan tienen una gran variedad de formatos predeterminados que pueden ser seleccionados según las necesidades del usuario o bien creados de forma personalizada, con la posibilidad de incluir imágenes y gráficas que permitan una mejor apreciación de los datos consultados. Cuenta entre sus principales funcionalidades con un módulo que permite realizar consultas SQL (Lenguaje Estructurado de Consultas) de forma gráfica, de gran

relevancia para las empresas que se valen de este tipo de herramientas ya que permite conocer el estado de las entidades que manejan y las relaciones que prevalecen entre estas.

A nivel mundial existe un gran cúmulo de sistemas que responden a la misma funcionalidad del GDR, profesionales, cada uno con sus peculiaridades pero de rendimiento estable y eficiente. Pueden encontrarse aplicaciones privativas (Active Reports, Crystal Reports) o de licencias libres (Agata Reports, Jasper Reports), de las cuales se han observado su comportamiento y características fundamentales para desarrollar la versión 2.0 de GDR y obtener una herramienta integral de alta calidad y que responda a los requerimientos empresariales de las instituciones del país. Concretamente se utilizó en el desarrollo de dicha versión el motor de generación de reportes Jasper Reports, que es una tecnología libre y además da solución a problemas que afectaban la anterior distribución del software. A partir de la inclusión del nuevo motor fue posible añadir nuevas funcionalidades que elevan sustancialmente el rendimiento de la aplicación.

De forma general, ambos sistemas presentan funcionalidades muy atractivas para el manejo, control y aprovechamiento de la información para la logística empresarial. Su vinculación a un portal de servicios web basado en una Arquitectura Orientada a Servicios (SOA), junto a otros productos que los complementen (las aplicaciones SaaS se conciben en lo fundamental para ser integradas a plataformas de planificación, monitorización y gestión de recursos) puede constituir una propuesta atractiva de comercialización que a la vez amplíe los horizontes de mercado de los productos del Departamento de Integración de Soluciones.

1.3 Arquitectura de Software, conceptos e inicios.

Según se recoge en múltiples bibliografías …“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 estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera…Sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores”…(2) A partir de este pronunciamiento, otros como Parnas, Brooks y Sharp realizaron planteamientos que ayudaron a la conceptualización de la Arquitectura de Software (AS), aunque no fue hasta la década de los 90 donde sienta sus bases como disciplina dentro de la rama de las ciencias informáticas, a través de la monolítica obra de Dewayne Perry y Alexander Wolf, pertenecientes a los laboratorios de AT&T Bell de New Jersey y a la Universidad de Colorado, respectivamente, titulada “Foundations for the study of software architecture”, fragmentos de la cual se citan a continuación:

"…El propósito de este documento es construir el fundamento para la arquitectura de software…presentamos un modelo para la arquitectura de software que consiste en tres componentes: elementos, forma y razón. Los elementos pueden ser de procesamiento, datos o conexión. La forma se define en términos de las propiedades y relaciones entre los elementos, es decir, restricciones operadas sobre ellos. La razón proporciona una base subyacente para la arquitectura en términos de las restricciones del sistema, que lo más frecuente es que se deriven de los requerimientos del sistema. Discutimos los componentes del modelo en el contexto tanto de la arquitectura como de los estilos arquitectónicos"… (3)

Como puede apreciarse, estas ideas siguen vigentes y en constante desarrollo aún después de más de una década de ser pronunciadas. Así lo refleja Carlos Billy Reynoso al expresar que…"Los autores prosiguen reseñando el progreso de las técnicas de diseño… los investigadores pusieron en claro que el diseño es una actividad separada de la implementación y que requiere notaciones, técnicas y herramientas especiales". (2)

En vistas a crear un concepto estándar que definiera a la AS, la IEEE en su recomendación Nº 1471-2000 la describe como… "la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución". (4)

Al llevar a cabo un análisis de estos conceptos, se concluye que la AS es una poderosa herramienta puesta en manos del equipo de desarrollo. La misma da la posibilidad de organizar el trabajo en función de la planificación necesaria para hacer un software de calidad, con sus puntos críticos bien definidos, a partir de los cuales comenzar el ciclo de desarrollo. Permite obtener un conocimiento previo sobre las interacciones y dependencias entre las partes del sistema, para hacer que funcione como un todo. Además, plantea una línea base que guía al grupo de trabajo a alcanzar objetivos comunes que den cumplimiento a los requerimientos impuestos por el cliente.

1.3.1 Metodología de desarrollo.

El uso de una metodología de software ayuda considerablemente a organizar el trabajo a realizar por parte de los ingenieros, incrementando la calidad final de los productos que demanda la satisfacción de los clientes. La selección de la metodología a emplear va en correspondencia con las condiciones bajo las cuales tiene lugar el ciclo de vida del software. Entre los factores que determinan qué tipo de metodología aplicar (robusta o ágil) se encuentran el tiempo del que se dispone para implementar las aplicaciones, el tamaño del equipo de desarrollo, la experiencia de los miembros que componen el equipo y el conocimiento previo que existe en la organización sobre la metodología que se piensa aplicar.

En el Departamento de Integración de Soluciones los sistemas informáticos se construyen mayoritariamente de forma modular, utilizando equipos de trabajo pequeños que laboran en períodos de corta duración. Estos elementos, en principio, favorecen la adopción de una metodología ágil. Sin embargo, el hecho de que los equipos de desarrollo estén compuestos en su mayoría por estudiantes o ingenieros con pocos años de experiencia laboral, que dominan en lo fundamental el Proceso Unificado de Rational (metodología robusta), pone en duda el uso de metodologías ágiles como XP.
Ante esta situación, se impone la necesidad de utilizar una metodología que aporte dinamismo y organización a equipos de desarrollo pequeños, que disponen de un breve tiempo para implementar sistemas y que no cuentan con una amplia experiencia. OpenUp es una metodología ágil basada en RUP, y al igual que este establece un proceso dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental. La misma persigue la obtención de resultados satisfactorios con un menor costo de tiempo de producción, generando un mínimo de elementos de documentación para cada proyecto. No obstante, los mencionados artefactos que se obtienen son lo suficientemente específicos como para que su contenido pueda ser interiorizado por un personal con conocimientos elementales de los temas que describen. Es por todas estas razones anteriormente expuestas que se decide establecer OpenUp como metodología de desarrollo.

OpenUp define cuatro fases específicas para el ciclo de vida de los proyectos:

Concepción: se determinan los objetivos y se establece el alcance del proyecto.

Elaboración: se establece la línea base de la arquitectura del sistema, creando las condiciones de estabilidad necesarias para el desarrollo de las siguientes fases.

Construcción: basado en la arquitectura y diseño definidos con anterioridad, se lleva a cabo la implementación y pruebas de los sistemas en desarrollo.

Transición: se realizan las tareas de comprobación pertinentes que garantizan el cumplimiento de todos los requisitos identificados con vista a la satisfacción de los acuerdos contraídos con los clientes.

OpenUp además define seis flujos de trabajo:

Requerimientos: se realizan reuniones con el cliente con el objetivo de comprender el funcionamiento de los procesos de su organización y definir los requerimientos del sistema informático a desarrollar.

Análisis y diseño: se refinan y diseñan los requisitos obtenidos para su posterior implementación.

Implementación: partiendo de los requisitos obtenidos se lleva a cabo la implementación de los mismos basándose en el diseño planteado.

Pruebas: se prueban los artefactos y componentes del sistema durante todo el ciclo de desarrollo en aras de identificar y corregir los errores o imperfecciones existentes.

Gestión de Proyecto: corresponde a la planificación y realización de actividades durante el ciclo de desarrollo que repercutan de forma positiva en la calidad del producto obtenido.

Gestión de Configuración y cambios: tiene como objetivo establecer un control sobre los elementos producidos por el equipo de trabajo atendiendo al uso, actualización y control de versiones de los mismos.

1.3.2 Estilos arquitectónicos.

El primer concepto enunciado para describir un estilo arquitectónico, aparece paralelo al surgimiento de la AS, dado por Perry y Wolf, el cual plantea: “una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles” (3). Posteriormente, Mary Shaw y Paul Clements, hacen referencia a un estilo arquitectónico como “un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo”… (5)

Teniendo como base la consulta de diversas fuentes bibliográficas, se concluye en que existen alrededor de seis clasificaciones para agrupar los estilos arquitectónicos difundidos, estas son:

1- Estilos de flujos de datos.
- ✓ Tuberías y filtros.

2- Estilos de invocación y retorno.
- ✓ Arquitectura en capas.
- ✓ Arquitectura basada en componentes.
- ✓ Modelo Vista Controlador.

3- Estilos jerárquicos.
- ✓ Cliente-servidor.

4- Estilos de código móvil.
- ✓ Máquina virtual.

5- Estilos peer-to-peer.

- ✓ Arquitecturas Orientadas a Servicios (SOA).

Tuberías y filtros.

Se encuentra entre los de más temprana definición, perteneciente al grupo de los estilos de flujo de datos. En su nombre el término “filtros”, que pudiera confundir con lo que en realidad representa, no actúa como tal, es decir, desechando información o registros, sino que realizan procesos de transformación sobre los datos que transitan por las tuberías. Por tanto, da lugar a la implementación de procesos secuenciales para el manejo de datos, donde las entradas van siendo modificadas en cada filtro, convirtiéndose progresivamente en las salidas del sistema. El autor Carlos Reynoso plantea que el uso de este estilo (6) se refleja en toda aplicación típica donde un cliente realiza una solicitud, la cual es validada para luego a través de un servicio web se realice una consulta a la base de datos y se devuelva la respuesta al cliente en formato HTML.

Entre las ventajas del estilo se encuentran su facilidad de comprensión e implementación, incluso en la realización de procesos complejos. Fuerza el desarrollo de procesos de forma secuencial, siendo fácil de envolver en transacciones atómicas. Además de esto, se puede llevar a cabo un empaquetamiento de filtros, para que trabajen de forma paralela o distribuida.

Independientemente de las antes mencionadas facilidades que presenta, la simplicidad de este estilo es algo que puede atentar en la implementación de servicios con una lógica de negocios complicada. También se señala que no maneja de forma óptima la lógica de control donde se haga uso de bucles o construcciones condicionales. El estilo tampoco da soporte a situaciones interactivas, como por ejemplo actualizaciones incrementales de información en pantalla.

Sin embargo, este estilo resulta de gran utilidad para un buen número de funcionalidades que son comunes en las aplicaciones, dejando para tareas más complicadas el uso de otros estilos de mayor profundidad.

Arquitectura en capas.

La arquitectura en capas es uno de los estilos que se utiliza con mayor frecuencia. Una definición bastante acertada de este estilo es la dada por Garlan y Shaw (7), la cual plantea que la arquitectura en capas forma una estructura jerárquica, en la que cada capa brinda servicios a su inmediata superior y consume de su inferior más próxima. Este funcionamiento se basa en ideas que se remontan a la década de los 60, pertenecientes a Edsger Dijkstra y de gran repercusión para la posteridad. Cada capa en sí puede contener múltiples subsistemas que integran recursos y servicios, dando un mayor grado de abstracción al sistema. En general, los procesos que se definen en cada capa y las

relaciones existentes entre las mismas derivan en un mejor dominio del funcionamiento del sistema en cuestión.

Se tienen como adversidades del uso de este estilo el hecho de que no es posible realizar un mapeo adecuado a estructuras jerárquicas en muchas situaciones, aunque por cuestiones de rendimiento sea necesaria la interacción directa de capas superiores con otras de mayor profundidad. Se añade también que si bien una arquitectura en capas ayuda al encapsulamiento y control en aplicaciones complejas, tiende a crear complicaciones innecesarias en sistemas sencillos.

Su utilización por otra parte trae consigo un grupo de beneficios que lo hacen relevante en el desarrollo de aplicaciones de gran envergadura. Típicamente en el departamento se construyen aplicaciones de gran solidez que integran numerosas funcionalidades. Por este motivo se puede aprovechar la abstracción que aporta a las aplicaciones este estilo en función de dividir requerimientos complejos en subsistemas que den solución al problema. Por otra parte al establecerse una estructura jerárquica es posible realizar cambios específicos atendiendo a las dependencias funcionales que existen entre las capas sin que se afecte el funcionamiento del sistema. A todo esto agregar que también facilita la aplicación de técnicas de reutilización de componentes.

Modelo Vista Controlador (MVC).

Según cita el autor Carlos Reynoso en Estilos y Patrones en la Estrategia de Arquitectura de Microsoft (6), el estilo arquitectónico Modelo Vista Controlador separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- ✓ Modelo: administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

- ✓ Vista: maneja la visualización de la información.

- ✓ Controlador: interpreta las acciones del ratón y del teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Este estilo está constituido de forma tal que la vista y el controlador dependen del modelo, no siendo así en el caso del último, que no tiene dependencias de las otras clases. Esto hace posible que se pueda construir y probar el modelo independientemente de la representación visual. En aplicaciones web es frecuente encontrar una separación bien definida entre la vista (browser) y el controlador (los componentes del lado del servidor que manejan los requerimientos HTTP).

Las ventajas que aporta este estilo son consistentes, dado que la vista se haya separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por otra parte, es bien conocido que los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de los negocios. Esto puede ser causado tanto como por las nuevas tendencias tecnológicas que se van introduciendo o con el mismo transcurso del tiempo y dado que el modelo no depende de las vistas, el agregar nuevas opciones de presentación no es algo que generalmente traiga implicaciones en el modelo.

Entre las desventajas que se han señalado con respecto al MVC, se encuentra el aumento en la complejidad que produce en las aplicaciones. Otro factor con una repercusión negativa observado se tiene en el costo de las actualizaciones frecuentes. Desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Si el modelo experimenta cambios frecuentes, podría ocurrir un desbordamiento de las vistas producido por un incremento significativo de requerimientos de actualización.

Existe una buena variedad de framework de desarrollo para la web que incluyen esta estructura antes referenciada que propone el MVC. Ejemplos claves de esto resultan Symfony y Zend Framework, en el caso de Symfony se hará un examen más exhaustivo del porqué de su selección por encima de los restantes framework, determinado además de otros factores por el alto grado de organización estructural que posee y la experiencia y respaldo con que se cuenta en la universidad en el trabajo con dicha herramienta.

Arquitecturas Orientadas a Servicios (SOA).

Las Arquitecturas Orientadas a Servicios (SOA), sentaron las bases del SaaS y representan un estilo arquitectónico que permite conectar varios sistemas empresariales que típicamente responden a una base de información común. Su función consiste en establecer reglas de control e integración a los procesos del negocio, haciendo más provechosa la explotación de la información.

A continuación se exponen concepciones de varios autores relacionados con el tema en aras de ampliar el concepto de SOA y ofrecer una idea más completa:

…“SOA es una arquitectura conceptual de negocios, donde la funcionalidad del negocio, o lógica de la aplicación, está disponible para los usuarios de SOA, o de los consumidores, como servicios reutilizables en una red informática. "Servicios" en una SOA son los módulos de negocio o funcionalidad de la aplicación con interfaces expuestas, y que son invocados por mensajes”… (8)

…“SOA establece un modelo de arquitectura que tiene como objetivo mejorar la eficiencia, agilidad y productividad de una empresa de servicios (…) La lógica de la solución queda representada en apoyo de la realización de los objetivos estratégicos asociados a la computación orientada a servicios”…(9)
Con la aplicación de este estilo se obtienen beneficios tales como la posibilidad de sustituir componentes individuales del sistema sin que afecten el funcionamiento de otros, y la facilidad de adaptar nuevos servicios. El sistema queda dotado de una arquitectura sencilla, robusta, homogénea y escalable que permite ganar en tiempos de implantación, operación y mantenimiento. Con la integración de la lógica del negocio también se logra un mejor procesamiento de la información por lo que se obtienen resultados más satisfactorios.
A pesar de todas esas ventajas antes mencionadas, SOA presenta factores en contra, entre los que se encuentran la velocidad de transmisión de los datos, que es menor en comparación con una conexión directa entre sistemas. Por otra parte, la implantación de una estructura tipo SOA suele ser un proceso lento dada la repercusión que tiene sobre sistemas en etapas de desarrollo. Sin embargo, es necesario destacar que con el tiempo los resultados que se obtienen de su correcta implementación contrarrestan en buena medida los recursos invertidos en dicha tarea, por lo que cada vez es más frecuente encontrar sistemas empresariales que hacen uso de este estilo en sus aplicaciones.

1.3.3 Patrones.

Los patrones surgen como resultado de probar una solución con éxito en múltiples ocasiones ante un tipo de problema determinado. Un patrón agrupa los conocimientos específicos obtenidos de la experiencia acumulada y de cuya aplicación depende la solución del agravante al que se le hace frente. En dependencia del tipo de problema al que dan solución, se clasifican en patrones de diseño, de arquitectura, entre otros. Para su definición se les asigna además un nombre, se especifica el problema que resuelven y los resultados que se obtienen con su aplicación.
Los patrones, a diferencia de los estilos arquitectónicos, se encuentran en grandes cantidades y con un alto grado de variabilidad (10). Producto de esto y lo difícil que resulta su revisión a la hora de desarrollar una aplicación, se recomienda utilizar aquellos que son afines a los estilos arquitectónicos que caracterizan la arquitectura sobre la que se trabaja. A continuación se presentan de forma general dos grandes grupos de amplia utilización en la Arquitectura de Software: los Patrones de Software para la Asignación General de Responsabilidades (GRASP, contenidos en la Tabla 1), y los patrones GOF (Gang of Four, contenidos en la Tabla 2). Algunos de estos, junto a otros que serán descritos posteriormente, formarán parte de la propuesta arquitectónica realizada en esta investigación.

Tabla 1: Patrones GRASP.

Primarios	Soporte
Experto	Variaciones Protegidas (Polimorfismo)
Creador	Fabricación Pura
Controlador	Indirección
Alta Cohesión	No hables con extraños
Bajo Acoplamiento	

Tabla 2: Ejemplos de Patrones GOF.

Creación	Estructurales	Comportamiento
Abstract Factory	Adapter	Command
Builder	Bridge	Iterator
Factory Method	Composite	Observer
Prototype	Decorator	State
Singleton	Facade	Visitor

1.3.4 Lenguajes de programación.

Entre los lenguajes de programación de mayor difusión utilizados en el mundo y en Cuba para hacer aplicaciones web (incluyendo aplicaciones tipo SaaS), gozan de la mayor aceptación PHP5, ASP, Java y Ruby, a continuación se hará una descripción de las cualidades de cada uno.

ASP .NET.

ASP es un lenguaje desarrollado por Microsoft, razón por lo cual utiliza tecnologías del tipo propietario. Es además un lenguaje compilado que cuenta con una buena velocidad de respuesta por parte del servidor, superior en cuanto a seguridad y robustez con respecto a su predecesor ASP. Destaca su excelente conectividad con el gestor de BD SQL Server (privativo también) e introduce el concepto de

“code-behind”, que separa la estructura de la página en dos ficheros, uno para la interfaz de usuario y el otro para el código. Otra de sus ventajas está en la capacidad que tiene de permitir el almacenamiento en la caché del servidor de variables e incluso páginas enteras, lo cual economiza el uso de recursos por parte del servidor en páginas con muchas consultas a BD.

Java.

Java es un lenguaje de programación de alto nivel, multiplataforma, muy poderoso, orientado a la POO (Programación Orientada a Objetos) y dinámico, ya que puede hacer llamadas a clases o recursos en tiempo de ejecución si estos son invocados. Cuenta con una curva de aprendizaje relativamente fácil de asimilar y una gran comunidad de expertos, soporte, información y seguimiento. Las tecnologías desarrolladas por Sun en su mayoría fueron liberadas entre 2006-2007 bajo licencias libres (GNU GPL), por lo que se pueden considerar como software libre, hasta la absorción de Sun por Oracle en 2010, a partir de este momento las herramientas creadas salen al mercado bajo licencias privativas de la propia Oracle.

Ruby.

Este es un lenguaje que en nuestros días está creando grandes expectativas. Es orientado a objetos, interpretado, corre en múltiples plataformas y se adquiere de forma gratuita, por lo que resulta atractivo para muchas organizaciones. Posee similitudes con otros lenguajes de programación como Perl y Python, por lo que programadores con conocimientos de estos lenguajes se adaptan con mayor facilidad al mismo. Su creador emplea el principio de la “menor sorpresa”, lo cual se traduce en un menor número de errores y simplicidad en el lenguaje. También goza de una buena conectividad con gestores de BD como PostgresSQL, MySQL y MS SQL Server.

PHP5.

Es un lenguaje de programación interpretado, orientado a la confección de páginas web dinámicas, y puede ser insertado en código HTML. Entre sus principales características están el ser un lenguaje multiplataforma (corre lo mismo sobre Windows que distribuciones de Linux) de uso gratuito. Goza de buena capacidad de conexión con múltiples gestores de bases de datos tales como PostgresSQL, MySQL, Oracle y MS SQL Server. Además de esto, puede utilizarse para realizar POO y soporta (a partir del propio PHP5) el trabajo con excepciones. Posee varias bibliotecas donde se describen y ejemplifican cada una de sus funciones, y cuenta con una amplia comunidad de programadores por la docilidad de su aprendizaje. Por este conjunto de características, además de la experiencia que poseen los programadores del departamento en el trabajo con PHP se considera dentro de las

propuestas como la más atractiva y por tanto se selecciona como lenguaje de programación de preferencia para la codificación de las aplicaciones a desarrollar por el departamento.

1.3.5 Framework.

Una buena definición de framework obtenida como resultado de la revisión de la bibliografía consultada, describe un framework como "una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado (…) puede incluir un soporte de programas, librerías y lenguajes de scripting (…) para ayudar a desarrollar y unir los diferentes componentes de un proyecto"… (11)

Para llevar a cabo la elección del framework de desarrollo a emplear, hay que tener en cuenta que esta decisión viene estrechamente relacionada al lenguaje de programación que se ha seleccionado para construir la aplicación. Para cada uno de los lenguajes anteriormente descritos, existen marcos de trabajo de preferencia por parte de los desarrolladores por las ventajas y comodidades que estos proporcionan en los distintos ámbitos de trabajo que enfrentan diariamente. A continuación se brinda una descripción de los framework más utilizados para el desarrollo de aplicaciones web y productos SaaS.

Struts.

Este es un framework mantenido por Apache Struts Project, destinado a la creación de aplicaciones web utilizando Java EE (Enterprise Edition). Propone una arquitectura basada en el estilo Modelo Vista Controlador (MVC). Es también perteneciente a la rama del software libre, lo cual hace junto a su integración con AJAX una propuesta atractiva para programadores que utilizan Java.

Spring.

Al igual que Struts es un framework para Java, orientado hacia el software libre. Propone una estructura para aplicaciones basada en el estilo arquitectónico MVC y soporta el trabajo con tecnologías modernas como AJAX y HTML5, también es fácil su integración para el desarrollo de aplicaciones con gestores de BD tradicionales.

.NET.

.NET es un framework robusto y seguro con un alto nivel de abstracción, desarrollado por Microsoft para el desarrollo de aplicaciones web. Destaca entre sus características la gran variedad de lenguajes que soporta, (lo cual le confiere una gran flexibilidad y hace que la curva de aprendizaje para su uso sea fácil de vencer) así como una amplia biblioteca provista de funciones para cada uno de estos. Es por definición, producto del trabajo de Microsoft, un software privativo.

Ruby on Rails.

Este es el framework más popular para el trabajo con Ruby. Es una herramienta libre y multiplataforma, que basa su arquitectura en el estilo MVC. Cuenta con una buena bibliografía mantenida por su creciente comunidad y trabaja con una buena cantidad de gestores de BD como PostgresSQL, MySQL, Oracle, entre otros.

Zend Framework.

Zend Framework es un framework de código abierto que tiene como principal patrocinador del proyecto a Zend Technologies, aunque cuenta con el soporte de numerosos colaboradores y empresas. Creado con el objetivo de implementar aplicaciones web haciendo uso de PHP5, es enteramente orientado a objetos (POO). Sus componentes se encuentran conectados débilmente (por lo que pueden ser utilizados de forma separada), aunque en conjunto forman un robusto marco de trabajo de gran rendimiento, valiéndose del estilo arquitectónico MVC y componentes para la presentación, filtrado y validación de formularios, que hacen más agradable el trabajo al desarrollador.

Symfony.

Symfony es un framework de la familia PHP, promovido bajo la licencia MIT de software libre. Multiplataforma, de fácil instalación y mantenimiento ya sea sobre Linux, Windows o Mac. Posee una gran flexibilidad y extensibilidad, cuenta con numerosos plugins y herramientas que posibilitan un mayor índice de reutilización de código. Se basa en el estilo MVC, estructura que permite una visualización mayor de los elementos que integran la aplicación, organizando el trabajo del programador y disminuyendo el tiempo de desarrollo. Se integra perfectamente con cualquier sistema de gestión de BD, brindando la posibilidad de sustituir uno por otro (Symfony es independiente del gestor de BD) en cualquier momento sin que se afecten en lo más mínimo las aplicaciones. Esto se debe a la utilización del ORM Propel, el cual viene por defecto con el framework, aunque también puede hacerse uso de Doctrine. Symfony es también el marco de trabajo más documentado del mundo, cuenta con miles de páginas distribuidas en libros y tutoriales en más de cuarenta idiomas, distribuidos de forma gratuita y con una amplia comunidad de desarrollo. Amparado además en la labor de la empresa Sensio Labs (de la cual Fabien Potencier, creador de Symfony, es CEO), encargada permanentemente de su estado y actualizaciones (las versiones estables difundidas se mantienen por espacio de 3 años sin cambios pero en constante proceso de perfeccionamiento), ya que basa su entrada de recursos a partir de las aplicaciones que desarrolla con dicho framework. El código fuente de Symfony posee ya más de ocho mil pruebas unitarias y funcionales, siendo utilizado para la creación de gigantescas aplicaciones web como Yahoo y Dailymotion, sumando entre ambas

cifras superiores a los doscientos cincuenta millones de usuarios, dejando más que claro el grado de escalabilidad que proporciona. Dadas las características anteriormente referenciadas y la solidez de dichos argumentos se selecciona Symfony como framework de desarrollo a emplear por los equipos de trabajo que conforman el departamento.

1.3.6 Servidor web.

IIS.

IIS (Internet Information Services) es el servidor web de Microsoft diseñado para trabajar con aplicaciones web. Es robusto y estable, con soporte para múltiples conexiones y lenguajes como ASP, ASP.NET, Perl y PHP. Constituye una herramienta de tipo propietaria.

Apache.

Apache es un servidor web de carácter libre, con la capacidad de correr en múltiples plataformas. Su arquitectura modular le confiere un alto grado de extensibilidad y configurabilidad, pudiendo incorporársele numerosos scripts con gran cantidad de funcionalidades que respondan ante posibles errores. Permite la elaboración y gestión de logs, que dan a los administradores una información completa del estado del servidor. Es compatible con una buena cantidad de lenguajes script como PHP, Perl, Java y documentos JSP, comportándose de forma estable ante aplicaciones dinámicas. Posee una documentación abundante proveniente tanto de la Apache Software Foundation como de la amplia comunidad que lo respalda. Por estas condiciones y la robustez que posee como herramienta para el control, mantenimiento y soporte de aplicaciones se escoge como servidor web y de aplicaciones a utilizar para el trabajo con las aplicaciones creadas por el departamento.

1.3.7 Servidor de aplicaciones.

Cuando se habla del servidor de aplicaciones a emplear, se debe tener en cuenta que las arquitecturas que incluyan entre sus tecnologías al servidor web Apache 2.0 o al IIS de Microsoft, ambos realizan una doble función desempeñándose como servidor web y servidor de aplicaciones al mismo tiempo. En el caso de que se utilice Rails para el desarrollo, lo más común es incluirle al Apache el módulo Passenger (a partir de la versión 3 de Passenger puede emplearse como servidor de aplicaciones independiente, aunque por determinados aspectos se recomienda mantenerlo funcionando junto a Apache). Si en lugar de utilizar Ruby se hace uso de Java, lo más común es utilizar como servidor de aplicaciones Tomcat. Este tiene como principal ventaja además de ser libre que, al estar desarrollado en Java, corre sobre cualquier sistema operativo que cuente con una máquina virtual de Java. Es estable, soportando volúmenes de carga elevados e implementa las tecnologías Java Servlet y JSP.

1.3.8 Servidor de Bases de Datos.

Un sistema de gestión de base de datos es un software que sirve como interfaz entre usuario, aplicación y base de datos. Basa su funcionamiento a partir de tres lenguajes: uno para la definición de los datos, el segundo para la manipulación de los mismos y el último para la realización de consultas. Su función consiste en facilitar el manejo, organización y cuidado de grandes cúmulos de información.

PostgresSQL.

PostgresSQL es un poderoso SGBD relacional y orientado a objetos, libre y multiplataforma, distribuido bajo la licencia BSD. Es capaz de realizar variadas y complejas operaciones, que van desde consultas de alto nivel, uso de triggers (disparadores), al soporte de consultas concurrentes (varios usuarios interactuando sobre una misma BD, donde el servidor muestra para cada una de las conexiones el resultado de sus operaciones sin que esto interfiera en el trabajo de otros consultantes; a esto se le conoce como control de versionado concurrente o MVCC por sus siglas en inglés). Posee un elevado grado de personalización, ya que permite la declaración de nuevos tipos de datos y funciones. También maneja el trabajo con lenguajes de programación como PHP, Perl, C++, Python, Ruby y C. Cuenta con una amplia comunidad de entusiastas a nivel mundial, que aporta abundante bibliografía y dispuesta a ayudar ante cualquier inquietud. Dado su carácter de herramienta libre, su alto grado de funcionalidad y rendimiento además de la experiencia acumulada por los equipos de trabajo del departamento con dicho gestor, se selecciona como el candidato de preferencia a emplear para el manejo de los datos de las aplicaciones desarrolladas dentro del departamento.

MySQL.

Al igual que PostgresSQL es una herramienta robusta para el manejo de datos. Es de fácil configuración y aprendizaje, multiplataforma y goza de gran popularidad producto de su rendimiento y estabilidad. Hasta su versión 5.1.49 es de carácter libre, no sucediendo así con liberaciones posteriores pertenecientes a Oracle, distribuidas bajo licencias privativas.

1.3.9 Herramientas CASE.

Constituyen herramientas de gran utilidad y su uso se refleja entre las buenas prácticas en el proceso de creación del software. Al hacer referencia a las mismas, Pressman plantea que “intervienen en la automatización de las actividades de gestión de los proyectos, gestionan todos los productos de los trabajos elaborados a través del proceso y ayudan a los ingenieros en el trabajo de análisis, diseño y codificación, reduciendo el tiempo y esfuerzo invertido para la realización de los proyectos y ganando en calidad los productos que se derivan de los mismos”. (12)

Rational Rose.

Rational Rose como herramienta para modelado es muy potente, es junto a Visual Paradigm de las favoritas en todos los entornos de desarrollo por su gran funcionalidad. Es multiplataforma, corre por igual en distribuciones de Windows o Linux, es además integrable con diferentes IDE (acrónimo en inglés de Ambiente Integrado de Desarrollo) y orientada a objetos. Realiza generación de código e ingeniería inversa para varios lenguajes de programación. Utiliza UML como lenguaje para el modelado y permite la exportación de diagramas en archivos de imágenes y XML.

Visual Paradigm.

Es una herramienta CASE de modelado muy completa, de gran utilidad durante todo el ciclo de desarrollo del software. Es multiplataforma, y puede integrarse con varios IDE, tales como NetBeans, Eclipse, JDeveloper y JBuilder, utiliza UML como lenguaje de modelado (soporta UML 2.1). Permite el uso de técnicas tales como el mapeo objeto-relacional (ORM), la generación de código e ingeniería inversa del mismo en varios lenguajes. Además exporta diagramas en formatos de imagen o XML y los proyectos realizados con Visual Paradigm pueden ser editados en otras herramientas CASE como Rational Rose. Se selecciona Visual Paradigm como herramienta de modelado a emplear por los equipos de desarrollo inherentes al departamento por el nivel de familiarización que tienen estos con el uso de dicha herramienta. Se emite este criterio teniendo en cuenta también el amplio espectro de solución a situaciones dentro del ámbito de desarrollo a las que da cobertura dicho software.

1.3.10 Lenguaje Unificado de Modelado (UML).

Es un lenguaje de modelado utilizado para visualizar, especificar, construir y documentar los elementos que componen un sistema. Cuenta con herramientas gráficas que permiten representar aplicaciones o proyectos haciendo uso de diagramas, teniendo en cuenta las reglas y relaciones que existen entre sus componentes; y cabe la posibilidad de que dichas aplicaciones puedan estar orientadas al trabajo con objetos. Es necesario destacar que independientemente de que su función es realizar descripciones de sistemas para facilitar el entendimiento de sus procesos, no especifica la forma en que se han de implementar los mismos.

1.4 La Calidad en la Arquitectura de Software.

El mundo de la informática y las comunicaciones es en extremo dinámico, en constante evolución y cambios. Las empresas de software en competencia tienen el reto de producir aplicaciones con un alto nivel de calidad, que satisfagan la demanda de sus clientes y de esta forma ganar en prestigio y posición en el mercado. Por tanto, es necesario conocer primeramente ¿Qué se entiende por Calidad

de Software? Según el concepto emitido por la IEEE (Institute of Electrical and Electronics Engineers), la Calidad de Software “es el grado en que el software posee una combinación deseada de factores”. Otro punto de vista, de preferencia para el autor y que gana en profundidad con respecto a la cita anterior, es la definición dada por Roger Pressman, el cual precisa que la Calidad de Software viene dada por “la concordancia con los requisitos funcionales y de rendimiento establecidos con los estándares de desarrollo explícitamente documentados y con las características que se espera de todo software desarrollado de forma profesional” (12). Pueden estudiarse numerosos y variados planteamientos dados por autores e instituciones reconocidas, mas todas las vertientes coinciden de forma general en que para que exista calidad, los requerimientos expresados en las necesidades del cliente y los atributos de calidad con que ha de cumplir todo software han de ir de la mano.

A la vez, se entienden por atributos de calidad a las propiedades de un sistema que permiten a los usuarios e interesados en el mismo juzgar el estado en que se encuentra. Estos atributos son influenciados significativamente por la arquitectura de software de cada sistema, debido a las decisiones que se toman en favor de potenciar los de mayor relevancia de acuerdo al tipo de aplicaciones que se pretende desarrollar. Dadas sus características, son agrupados en dos clasificaciones:

- ✓ Observables vía ejecución: como su nombre lo indica, son aquellos que pueden apreciarse a través del comportamiento del sistema en tiempo de ejecución (Tabla 3).

- ✓ No observables vía ejecución: a esta rama pertenecen los atributos que se establecen durante la construcción del sistema. (Tabla 4).

Tabla 3: Ejemplos de atributos de calidad observables en tiempo de ejecución.

Atributos de calidad	Descripción
Disponibilidad	Es la medida en que un sistema es accesible para su uso.
Confidencialidad	Se trata como la ausencia de acceso no autorizado a la información.
Desempeño	Grado en que el sistema cumple con sus funciones, ateniéndose a ciertas condiciones de velocidad, consumo de memoria, entre otras.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.

Confiabilidad	Da la medida de la robustez de un sistema para mantenerse operando satisfactoriamente a lo largo del tiempo.
Seguridad	Medida en que no se producen fallos o pérdidas de información en el sistema, a la vez que este es capaz de resistir ataques o usos indebidos mientras sirve a usuarios legítimos.

Tabla 4: Ejemplos de atributos de calidad no observables en tiempo de ejecución.

Atributos de calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto de realizar cambios en el sistema.
Mantenibilidad	Capacidad de modificar el sistema de manera rápida y a bajo costo.
Interoperabilidad	Medida en que partes de un sistema pueden trabajar con otro sistema externo.
Portabilidad	Habilidad del sistema para ser ejecutado en diferentes ambientes computacionales (de hardware, software o ambos).
Reusabilidad	Capacidad de diseñar un sistema de forma tal que parte de su estructura o componentes puedan ser reutilizados en futuras aplicaciones.

Dominar el campo del conocimiento referente a los atributos de calidad resulta indispensable para la realización de pruebas a una arquitectura. El llevar a cabo dichas pruebas, permite conocer qué tan presentes están los factores que han sido priorizados, así como información valiosa sobre los riesgos y fortalezas existentes en la arquitectura. Estos datos permiten poner en práctica las variantes arquitectónicas que promuevan el mayor nivel de aseguramiento posible que se espera obtener de cada uno de los atributos que determinan la calidad del sistema. Como resultado, además, mediante la observación de las relaciones y dependencias que existen entre dichos elementos, el arquitecto puede establecer un equilibrio en la medida que desee que el sistema posea cada una de estas cualidades.

Hasta el momento, han sido expuestos los motivos que propician el desarrollo de las pruebas de arquitectura y las variables sobre las que estas operan. En este proceso, intervienen también otros elementos (díganse técnicas y métodos) que permiten su conducción de manera exitosa. Tales prácticas, de estrecha vinculación a los temas aquí tratados, serán descritas en los epígrafes siguientes.

1.4.1 Técnicas de evaluación de arquitecturas de software.

En principio, se plantean cuatro técnicas fundamentales para la evaluación de arquitecturas de software, basadas en enfoques particulares, que son expuestas a continuación:

- ✓ Evaluación basada en simulación: consiste en la implementación de componentes de la arquitectura y del contexto del sistema donde se supone va a ejecutarse con un determinado nivel de abstracción, con la finalidad de evaluar el comportamiento de la arquitectura bajo ciertas circunstancias. Como instrumentos asociados a las técnicas de evaluación basadas en simulación, se tienen los lenguajes de descripción arquitectónica (ADL, abreviado en inglés) y los modelos de cola.

- ✓ Evaluación basada en modelos matemáticos: intervienen en la evaluación atributos de calidad operacionales. Permiten evaluar estáticamente el diseño arquitectónico y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Se destacan dos instrumentos fundamentales: las Cadenas de Markov y los Diagramas en Bloques de Fiabilidad.

- ✓ Evaluación basada en experiencia: se realiza una evaluación informal, que es llevada a cabo por los arquitectos de software durante el proceso de diseño y/o por equipos externos de evaluación de arquitecturas. Ambos casos giran en torno a la aglutinación de conocimientos del dominio que ayudan a justificar decisiones de diseño considerando las experiencias acumuladas. Existen tres instrumentos aplicables en esta técnica: intuición-experiencia, tradición y proyectos similares.

- ✓ Evaluación basada en escenarios: plantea Kazman que un escenario está compuesto por tres partes: estímulo, contexto y respuesta, y que proveen un vehículo que permite concretar y entender los atributos de calidad. El estímulo es la parte del escenario que explica lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta muestra, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Los métodos que utilizan esta técnica cuentan con dos instrumentos de evaluación relevantes: el Árbol de Utilidades y los Perfiles propuestos por Kazman y Bosch respectivamente (13).

La técnica de evaluación basada en escenarios no necesita disponer de cuantiosos recursos para ser aplicada, por lo que resulta de fácil utilización, sin dejar de ser por esto eficiente en alguna medida. Provee una excelente base sobre la cual se han desarrollado robustos métodos de evaluación para

realizar mediciones a las arquitecturas de software. El Árbol de Utilidades, herramienta perteneciente a los métodos que implementan esta técnica, permite definir y detallar los atributos de calidad para cada arquitectura en particular.

Kazman plantea que el Árbol de Utilidad se construye a modo de esquema y en forma de árbol, en el que se presentan los atributos de calidad vinculados al software en cuestión (13). Estos atributos son refinados mediantes escenarios a través de los cuales se puede determinar el nivel de prioridad correspondiente a cada uno. Para cada atributo a su vez, se construyen una serie de escenarios de utilidad para realizar las mediciones a la arquitectura, con una escala de importancia y dificultad asociada.

1.4.2 Métodos de evaluación de arquitecturas de software.

Seguidamente se hace una presentación de los métodos de evaluación de arquitecturas basados en escenarios considerados de mayor relevancia en investigaciones realizadas con anterioridad, y por tanto candidatos potenciales para la realización de las pruebas a la arquitectura presentada.

Método de Análisis de Arquitecturas de Software (SAAM).

El Método de Análisis de Arquitecturas de Software (al inglés Software Architecture Analysis Method, SAAM) ha sido ampliamente difundido y documentado; siendo aplicable a la medición de variados atributos de calidad como son: portabilidad, integrabilidad, escalabilidad y modificabilidad. Su funcionamiento consiste en la organización de un conjunto de escenarios que recogen los posibles cambios o situaciones a los que pueden encontrarse expuestos los sistemas en el futuro. Para su uso, es necesario contar primeramente con una descripción de la arquitectura a probar. Como resultado devuelve los datos que arroja la proyección de los escenarios sobre la arquitectura evaluada. Otros beneficios posibles de obtener con su aplicación son arquitecturas alternativas o elementos de las mismas que contribuyan al mejoramiento de la arquitectura candidata.

Método de Análisis de Acuerdos de Arquitectura (ATAM).

El accionar del Método de Análisis de Acuerdos de Arquitectura (ATAM, Architecture Trade-off Analysis Method) está inspirado en el estudio de tres aéreas principales: atributos de calidad, estilos arquitectónicos y el método SAAM. Para aplicar el método se ha de partir de la identificación de los estilos o enfoques arquitectónicos que caracterizan la propuesta que se desea evaluar. Dado que estos constituyen las vías para alcanzar los atributos de calidad esperados, su análisis da la medida en que el sistema es capaz de ampliarse, integrarse con otros sistemas externos, responder ante cambios, además de otras circunstancias.

1.5 Conclusiones del capítulo.

Con el desarrollo de este capítulo se realizó un estudio del estado del arte y conceptos relativos a los modelos de negocio, haciendo énfasis en el modelo SaaS. Se presentó una panorámica del enfoque de mercado de algunos de los principales productos creados por el departamento y la relevancia que tendría el modelo de negocio planteado como nueva vía de comercialización del software a través de la web. Además de esto se llevó a cabo un estudio de los elementos de la Arquitectura de Software y tecnologías asociadas a los mismos que son utilizados a nivel mundial para la implementación de aplicaciones web enriquecidas, y que responden a las características del SaaS. Por último se hizo una selección de las que se consideran de mayor potencialidad teniendo en cuenta criterios de rendimiento, experiencia acumulada por los equipos de trabajo del Departamento y que defiendan los principios de soberanía tecnológica en el software.

Capítulo 2: Propuesta de Arquitectura.

En este capítulo se presenta la propuesta de Arquitectura de Software para el desarrollo de aplicaciones SaaS en el departamento, haciendo uso del expediente establecido en la universidad para el diseño de las arquitecturas de sus proyectos.

2.1 Consideraciones generales.

Se utilizarán como modelo para la descripción de las vistas de la arquitectura las contenidas en la Guía base metodológica para el establecimiento de Arquitecturas en los Proyectos UCI (Figura 1, Portal de Producción UCI, 2012). La misma se encuentra inspirada en el modelo CMMI (Integración de Modelos de Madurez de Capacidades: establecido para evaluar y optimizar los procesos asociados al desarrollo de software), del cual la universidad aspira a obtener el nivel dos una vez que concluya el proceso de mejoras por el que atraviesa a la fecha. En la guía se recogen nueve vistas, que deben incluir las arquitecturas candidatas presentadas por los proyectos en sus expedientes. En el caso particular de este trabajo de diploma, el cual responde a un modelo de negocio que plantea el uso de una estructura diferente a la utilizada en los productos desarrollados por el departamento, se tomaron las consideraciones siguientes:

- ✓ De las nueves vistas generales, en las correspondientes a procesos, presentación, sistema, datos y la vista tecnológica, se harán breves referencias en los puntos donde se entienda que tengan un impacto sobre la estructura general de las aplicaciones desarrolladas por el departamento a partir del modelo propuesto.

- ✓ En el caso de las vistas de seguridad, integración, infraestructura y despliegue constituirán el centro del trabajo de la propuesta de arquitectura que se presenta. Los factores que han devenido en esta decisión van en correspondencia con el hecho de que las primeras cinco vistas mencionadas no difieren en gran medida de un software o aplicación a otro; simplemente se ajustan a las características propias que dan solución al problema por el que fueron implementadas. Luego, sobre el segundo grupo de vistas mencionadas, influyen un conjunto de consideraciones arquitectónicas referentes a la seguridad, rendimiento e infraestructura de soporte, que guardan una estrecha relación entre sí. Estas serán desglosadas en los epígrafes siguientes que conforman este capítulo.

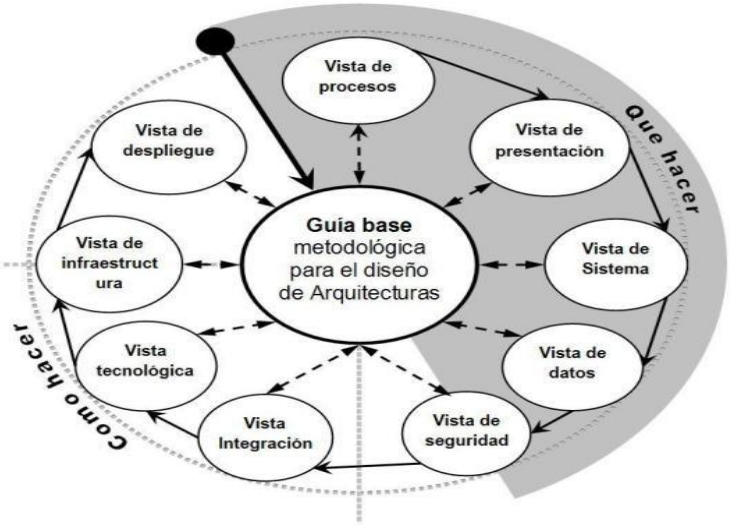


Figura 1: Guía base metodológica para el diseño de arquitecturas.

Por otra parte, no se tiene concebido que las aplicaciones desarrolladas por el departamento que implementen el modelo de negocio SaaS, se emprendan como proyectos aislados. El fortalecimiento del modelo viene dado por la posibilidad de que los usuarios puedan escoger entre los servicios ofrecidos aquellos que consideren vitales para la automatización de sus procesos. En tal sentido se hace necesaria la construcción de un portal web en el cual colocar cada una de las aplicaciones ya listas para su explotación. Se ha de presentar este portal de forma tal que cualquier usuario pueda tener acceso a su dirección electrónica, encontrando en el mismo una descripción detallada de la utilidad de uso de cada aplicación, siendo obligatorio autenticarse si desea contratar alguno(s) de los servicios (aplicaciones) allí expuestos. Para la contratación de servicios ha de establecerse una funcionalidad que permita la realización de dicha actividad, se recomienda el uso de una pasarela de pago como medio de liquidación para los acuerdos contraídos por el cliente para con su proveedor de servicios. Dicha pasarela, desde luego, debe proveer un vínculo a una entidad bancaria donde la empresa comercializadora ALBET tenga una cuenta habilitada, haciendo las transacciones seguras tanto para los clientes como para la empresa.

El desarrollo tanto de dicho portal como de las aplicaciones que aglutine, debe atenerse a los criterios de la Arquitectura Orientada a Servicios (SOA). Partiendo de las pautas que este estilo arquitectónico establece, será posible para el usuario acceder a información proveniente de otra aplicación distinta a la que él se halle haciendo uso en ese instante de tiempo sin tener necesidad de estar autenticado en la última (mas debe estar subscrito con anterioridad, lógicamente). En la implementación de este estilo sería provechosa la utilización del propio portal como “mediador” para el tráfico de la información entre las aplicaciones. Esta situación puede tomarse como puente para incitar a los clientes en favor de

contratar nuevos servicios teniendo como pretexto la obtención de nuevas facilidades que hagan más extensivo el uso de la información que maneja. Un argumento adicional sería la favorable repercusión que tendría en la toma de decisiones empresarial de su organización. Los criterios aquí expuestos y otros sobre la necesidad de la inclusión del estilo arquitectónico mencionado fueron abordados con mayor profundidad en el capítulo introductorio.

2.2 Vista de seguridad.

La seguridad constituye un factor apremiante dentro de cualquier sistema informático. De su correcta estandarización depende la integridad de los sistemas en uso y la información que estos manejan. El propósito de la vista de seguridad está dado por la búsqueda de las políticas y buenas prácticas que han de establecerse a lo largo del ciclo de vida del software a fin de mitigar o minimizar debilidades que puedan amenazar la calidad final de los productos a implementar (para un estudio más profundo del tema, el lector puede remitirse al artefacto correspondiente a la Vista de seguridad).

2.2.1 Autentificación, autorización, administración de roles y control de acceso con Symfony.

Cuando de seguridad se trata, existen conceptos que son imprescindibles dada su estrecha relación con el tema. Con la implementación de los procesos de **autentificación** y **autorización**, se obliga al usuario a presentar una “credencial” que lo identifique ante el sistema (autentificación). Una vez que el sistema verifica la autenticidad de dicha credencial (típicamente usuario y contraseña), procede a la comprobación de los privilegios que posee el usuario para determinar la concesión o no de los recursos que solicita (autorización).

Figura 2: Autentificación y autorización con Symfony.

El framework Symfony posee varias vías a través de las que pueden llevarse a cabo ambos procesos. Symfony implementa estos mecanismos permitiendo el acceso a recursos que no requieran de una identificación previa o activando un “cortafuegos” (autentificación) en caso de ser solicitados recursos que involucren permisos especiales para ser consumidos. Además de esto, tiene la capacidad de establecer una estructura de roles jerárquica, otro factor de suma importancia en materia de seguridad. Con el establecimiento de roles, se pueden definir los permisos con que debe contar cada usuario según el rol asignado a su perfil. Esta es otra medida preventiva que ha de implantarse en cada sistema, debido a que únicamente los usuarios con rol de administrador deben tener control total sobre las aplicaciones a las que tengan acceso; reservándose el manejo de las funcionalidades necesarias a los restantes roles según las responsabilidades que tengan asignadas en el sistema. Un ejemplo de esto se tiene en la típica funcionalidad de revisión de trazas que incluyen las plataformas SaaS. En su caso, sólo el administrador del sistema debe tener acceso autorizado para chequear las actividades y modificaciones realizadas sobre el software por los usuarios del mismo.

Otros conceptos convergentes a este tema son la **integridad** y el **no-repudio.** El término “integridad” se usa haciendo referencia a la garantía dada a que la información que se transmita entre dos entidades no sea intervenida por un tercero con objetivos malintencionados para con la misma. El “no repudio”, se basa en la verificación de la emisión de la información de un ente a otro, de forma tal que el emisor obtenga una prueba de la entrega de la información transmitida y el receptor del origen de transmisión de los datos. Para dotar el tráfico de información sensible de la protección adecuada, normalmente se hace uso de los protocolos criptográficos SSL/TSL, embebidos en el protocolo HTTPS. En las aplicaciones a desarrollar ha de hacerse uso intensivo de estas tecnologías atendiendo a la responsabilidad que se asume como proveedor de servicios con respecto a la seguridad e integridad de los datos depositados por los clientes. De esta forma el resultado de una intervención a la información transmitida, sería una secuencia de caracteres encriptados de poca/nula utilidad para el atacante.

2.2.2 Configuración del servidor web y de aplicaciones Apache.

La correcta configuración del servidor web y de aplicaciones constituye un importante eslabón dentro de la cadena de seguridad de los sistemas web. Combinando el uso de buenas prácticas, restricciones y la aplicación de parches de seguridad actualizados contribuirán de forma positiva a la disminución de vulnerabilidades ante posibles ataques y fallos del sistema. A continuación se muestran un conjunto de acciones a emprender para el correcto funcionamiento del mismo:

Restricción de acceso por IP: una de las características que más destacan a Apache es su alta configurabilidad. Por esto, es posible restringir el acceso desde direcciones de red no deseadas, otorgando permiso solo a aquellas que se utilizan para su administración y mantenimiento.

Restricción de acceso a archivos: deben limitarse los permisos de acceso de forma tal que solo se acceda a los directorios necesarios, según las funciones de los perfiles de administración establecidos.

Hacer uso de cuentas y grupos de usuario propios: existen versiones de Apache que poseen usuarios definidos por defecto, lo correcto es desactivar o desechar estos y establecer unos propios para evitar posibles ataques que se valgan de esta situación.

Disminuir los tiempos de espera: habitualmente Apache tiene establecido por defecto 300 segundos como tiempo de espera máxima. La disminución del valor de esta variable es una buena práctica a realizar con el fin de reducir factores de riesgo en los sistemas.

Establecer un límite para el tamaño de las peticiones: Apache cuenta con directivas que le permiten regular el tamaño de las peticiones que le llegan. Colocar un máximo adecuado en este elemento de configuración protege al servidor de sobrecargas que puedan hacerlo colapsar.

Desactivar la exploración de directorios: es posible evitar exploraciones no deseadas sobre el directorio de archivos haciendo uso de directivas en la etiqueta directorio.

Desactivar módulos innecesarios: normalmente existen una gran cantidad de módulos que pueden ser integrados a los servidores web. Teniendo conocimiento sobre el funcionamiento de estos, se puede controlar cuales deben mantenerse funcionando y deshabilitar aquellos que no aportan algo significativo de acuerdo a la configuración y servicios a los que se da soporte.

Ocultar información delicada: al instalarse Apache, por defecto suele mostrarse información considerada sensible acerca de este (versión, sistema operativo sobre el cual corre, módulos que se encuentran instalados). Esta información puede ser utilizada por terceros para realizar ataques al servidor. Configurando adecuadamente las directivas **ServerSignature Off** y **ServerTokens Prod** en el archivo **httpd.conf** evita el desencadenamiento de este tipo de incidentes.

2.2.3 Configuración del servidor de bases de datos.

La seguridad de los datos es un elemento sumamente delicado, ya que de su integridad y disponibilidad depende toda la lógica de negocio empresarial. Al hablar de las vías a utilizar para garantizar su protección, se trabaja sobre cuatro niveles de seguridad básicos.

Seguridad sobre el acceso al sistema: puede garantizarse mediante dos mecanismos. El primero basa su funcionamiento en la seguridad del sistema operativo sobre el cual corre el servidor, sirviéndose del apoyo que da la seguridad de acceso al sistema como prueba del acceso protegido a los datos almacenados. El segundo método deja a manos del propio servidor el control del acceso a los datos a través de la creación de cuentas de usuarios, asignando en cada caso las responsabilidades pertinentes.

Seguridad a nivel de objetos: se encarga de definir el nivel de operatividad de los usuarios sobre las bases de datos. Establece los permisos que tendrán estos para modificar, crear, eliminar los elementos que conforman la estructura de una base de datos: dígase tablas, relaciones, reglas, entre otros.

Seguridad a nivel de datos: la seguridad se maneja en la capa de información, señalando para cada usuario el nivel de acceso a la información y los permisos otorgados para su manejo.

Seguridad en la protección física de la información almacenada: aquí entran en juego las herramientas del sistema operativo encargadas del establecimiento de copias de seguridad y puntos de recuperación. Interviene también el uso de dispositivos de almacenamiento externos y servidores de réplicas como medios adicionales de prevención y protección de la información.

Así mismo, existen principios considerados básicos a tener en cuenta para el aseguramiento de la información en servidores de bases de datos. A continuación se exponen varios considerados fundamentales:

Disgregación de responsabilidades: tanto el administrador del sistema (ASO) como el de bases de datos (ABD) deben ser personas de confianza (ambas responsabilidades no deben recaer en el mismo individuo). De igual manera no deben contar con los mismos privilegios, han de hacer uso por separado de cuentas con usuarios y contraseñas diferentes en correspondencia con el rol que ocupan.

Mínimo de privilegios: se ha de realizar solo la instalación de software de servidores necesaria, deshabilitando todos aquellos servicios y puertos en desuso o de funcionamiento irrelevante para los

propósitos con que se tiene concebido el sistema. Las cuentas de administración solo deben estar en manos del personal requerido.

Restringir el acceso desde la red: el uso de un cortafuego es algo implícito dentro de las medidas para proteger un servidor ante posibles ataques. El cortafuego debe configurarse para que solo permita el acceso al servidor desde direcciones locales o estaciones específicas que no se encuentren en su subred. Por su parte, PostgreSQL a través de los ficheros **postgresql.conf** y **pg_hba.conf** se configura para indicar la forma en que se implementará la seguridad de las conexiones al sistema, especificando cuales usuarios, con que permisos, desde que direcciones IP y a que registros podrán acceder en las bases de datos. Tanto para las conexiones que se realicen desde la propia subred del servidor como para las externas que se hayan autorizado, se hará uso del método de encriptación md5 (configuración recomendada por PostgresSQL) para el cifrado del flujo de datos.

Uso de contraseñas fuertes: se estipula que las contraseñas a emplear para la administración de las bases de datos han de estar compuestas como mínimo por siete caracteres. Las mismas han de ser diferentes del usuario al que pertenecen y han de incluir en su composición letras en mayúsculas, números y caracteres especiales. El período de duración de una contraseña no excederá los dos meses.

2.2.4 Configuración de la herramienta de control de versiones Subversion.

El uso combinado de Subversion como herramienta para el control de versiones y el cliente RapidSVN permite a los equipos de desarrollo mantener un juego de salvas estable del código fuente, artefactos y aplicaciones que se generan como resultado de su trabajo.

Para la adecuada explotación de esta herramienta se tiene como práctica correcta la creación de roles que agrupen a los usuarios que van a hacer uso de la misma. Los permisos se asignan según la dependencia del papel que desempeñen cada uno de los roles dentro del ambiente de desarrollo. Entre los roles comunes que se definen con frecuencia para el manejo de la herramienta se hallan: jefe de línea, líder de proyecto, analista, arquitecto principal, gestor de calidad, planificador y gestor de configuración, entre otros.

2.2.5 Salvas de seguridad.

Realizar copias de seguridad de la información contenida en el servidor de control de versiones contribuye en gran medida a salvaguardar la misma de incidentes inesperados que atenten contra su integridad. Por tal motivo se deben mantener copias de la base de datos realizadas diariamente en un

servidor independiente destinado a estas funciones. Como medida adicional de protección, otra copia de las salvas puede almacenarse en una estación de trabajo debidamente asegurada y atendida por el encargado para esta responsabilidad dentro del equipo de desarrollo.

Como medidas complementarias se plantean:

- ✓ Realizar salvas dos veces por día del repositorio y la base de datos (configuradas de forma automática o realizadas manualmente), la primera a las 6:00pm y la segunda a la 1:30am.
- ✓ Transferir copias de estas salvas a una unidad de almacenamiento externa (disco curo).
- ✓ Mantener las salvas por un período de una semana.
- ✓ Las versiones estables de las aplicaciones desarrolladas se almacenarán en directorios independientes bajo acceso restringido.
- ✓ En el caso de las aplicaciones puestas en explotación, realizar salvas de sus bases de datos como mínimo dos veces al día.

2.2.6 La seguridad en el ciclo de desarrollo de las aplicaciones.

Otros aspectos a considerar durante el ciclo de desarrollo del software los constituyen el uso de estándares de codificación, validación y la ofuscación del código fuente (valiéndose de un software que sirva a este propósito). El establecimiento de políticas que dicten la forma de escribir las aplicaciones, no solo contribuye a una mejor comprensión del código por parte del equipo de desarrollo. También protege los programas escritos a través de la utilización de las funciones adecuadas del lenguaje que garanticen la seguridad desde el código fuente, entre otras facilidades. La parametrización en los componentes de validación es otro factor de peso, teniendo en cuenta que a partir de su adecuada implementación se asegura la entrada de datos a las aplicaciones en los formatos correctos. Estas funciones además previenen del bombardeo de consultas SQL a las bases de datos. El llevar a cabo la ofuscación del código por su parte, protege el software de ataques de ingeniería inversa. Con regularidad también al aplicar esta técnica sobre el código fuente de un programa, se obtiene un código de menor tamaño al eliminarse información redundante, haciendo que gane en velocidad de carga y ejecución.

A modo de resumen, puede decirse que la conjugación de estos factores tiene un impacto positivo en la calidad final de los productos. De igual forma sucede con la seguridad, estabilidad y rendimiento que se espera que tengan.

2.3 Vista de infraestructura.

La calidad del trabajo de los equipos de ingenieros informáticos depende desde etapas tempranas de la infraestructura con que cuenten para el desarrollo. Existe todo un engranaje complejo, invisible muchas ocasiones a la vista del usuario final, que responde al buen funcionamiento y perdurabilidad de los sistemas en explotación. El uso de las herramientas adecuadas, servicios en paralelo destinados a la gestión de proyecto y la implantación de sistemas de monitoreo en tiempo real, son algunas de las claves que aseguran la eficiencia, eficacia y rentabilidad que se pretenden ver materializados a lo largo de todo el ciclo de vida de un software (para un estudio más profundo del tema, el lector puede remitirse al artefacto correspondiente a la Vista de infraestructura).

Teniendo como punto de partida la bibliografía consultada y las características de las aplicaciones que se esperan obtener como resultado de la implantación de la arquitectura propuesta, se definen dos áreas fundamentales sobre las cuales sentar bases para la construcción de la presente vista de infraestructura: el entorno de desarrollo y el área de despliegue.

Entorno de desarrollo: es la base de la infraestructura que da soporte al proyecto. Aquí se definen las tecnologías y herramientas a emplear para el desarrollo del software y aquellas utilizadas en actividades concurrentes que garantizan la correcta realización de dicho proceso. Se hace lugar además para el establecimiento de políticas y estrategias a considerar con respecto al mantenimiento y operatividad de la plataforma de servicios.

Área de despliegue: constituye un área de gran relevancia para la infraestructura. En ella se llevan a cabo las pruebas a la arquitectura, con el objetivo de definir los requerimientos técnicos y de hardware sobre los cuales han de funcionar los sistemas. Intervienen también un conjunto de herramientas especializadas en el monitoreo y control de los recursos críticos utilizados (ancho de banda, RAM, procesadores, entre otros). Complementariamente, se planifican las estrategias con vistas al aumento paulatino de la escalabilidad y otros medidores de rendimiento con que ha de cumplir esta variante de plataforma de altas prestaciones.

2.3.1 Entorno de desarrollo.

Para el establecimiento del entorno de desarrollo se recomienda la inclusión en la mayoría de las posibilidades de herramientas y tecnologías libres, de acuerdo con las políticas de migración hacia el software libre establecidas en nuestro país y los criterios de selección planteados previamente. A la vez, la existencia de servicios de mensajería instantánea y correo electrónico como parte de la infraestructura de la universidad, resulta provechosa (señalada por Microsoft como necesaria para los

entornos de desarrollo) dado su valor como elemento para fomentar la comunicación entre los miembros del(los) equipo(s) de trabajo. El estudio realizado en el capítulo anterior sirve de base para el establecimiento de las herramientas que conformarán dicho entorno, junto a otras por definir que lo complementen.

Las herramientas que componen un entorno de desarrollo se dividen en dos grupos: horizontales y verticales. Las primeras son consideradas de propósito general y dan soporte y aseguramiento al proceso de desarrollo, dígase sistemas operativos, aplicaciones para el control de versiones, gestión de proyectos y gestión de la información, entre otras. Las herramientas verticales por su parte responden al tema específico del desarrollo de software. Como representación de las que se incluyen en este último grupo se tienen las herramientas de modelado, el IDE y el SGBD. A continuación se ofrecerá una presentación más detallada de ambas clasificaciones.

Herramientas horizontales.

Sistema operativo.

Se plantea como sistema operativo a utilizar por los equipos de desarrollo, en los servidores web, de aplicaciones y bases de datos en acuerdo con las políticas institucionales y en aras de la soberanía tecnológica, el uso de la versión 6.0 de Debian. Como base del planteamiento se tienen sus cualidades ya probadas en cuanto a estabilidad, facilidad de actualización, uso de memoria, tiempo de respuesta a las peticiones e implementación de la seguridad.

Herramienta para el Control de versiones.

Con el objetivo de realizar salvas del trabajo que se lleva a cabo se establece Subversion 1.6.12 como servidor de control de versiones y dado el sistema operativo seleccionado se empleará el RapidSVN en su versión 0.12.0 en las estaciones clientes. Con esto se protege el trabajo que se realice en caso de fallos inesperados o modificaciones erróneas sobre el código, las vistas u otro factor perteneciente a las aplicaciones, dando la posibilidad de establecer puntos de recuperación según la configuración con que se administre el software.

Documentación.

En el manejo de la documentación intervendrán para PHP PHPDoc y para JS JSDoc. Para la confección de la ayuda, tutoriales y manuales de usuario se hará uso de Libre Office, exportándose los mismos en formatos doc, pdf y html.

Herramientas verticales.

Herramienta de modelado.

Por su alta utilidad a lo largo de todo el ciclo de desarrollo de un software y la cantidad sustancial de problemas a los que da solución durante dicho proceso se hará uso de Visual Paradigm 8.0 como herramienta para el modelado. Otros factores determinantes que llevaron a esta decisión se encuentran expuestos de forma más extensa en el capítulo 1 del presente documento.

Entorno integrado de desarrollo (IDE).

NetBeans 7.1 es un potente IDE de código abierto y libre. Cuenta con facilidades ya incluidas desde versiones anteriores como son el soporte para el desarrollo de aplicaciones de tipo SOA, incluyendo herramientas para la confección de esquemas XML y editores de WSDL para el trabajo con servicios web. Posee un paquete visual para la web permitiendo crear aplicaciones web estándar con soporte para AJAX y componentes JSF incluidos. Desde luego, NetBeans puede utilizarse para crear aplicaciones escritas con PHP5, contando con un debugger integrado y soporte para el framework Symfony que hace uso del estilo arquitectónico MVC. La versión recomendada cuenta además con soporte para CCS3 y HTML5, incluye el uso de refactorización para PHP y mejoras en el debug para dicho lenguaje.

Frameworks.

Como parte de la selección de las herramientas y tecnologías se definió el uso de Symfony como framework para el desarrollo de aplicaciones, particularmente se propone la versión 1.4 de Symfony, aunque es de conocimiento la existencia de la versión 2.0. Para argumentar el porqué mantener el uso de la versión 1.4, comprobada ya su estabilidad por parte de los equipos de trabajo del departamento, se tiene como factor fundamental el costo que supondría la migración de las aplicaciones ya desarrolladas con dicha versión de la herramienta hacia la 2.0.

Las pruebas unitarias se realizarán haciendo uso de la biblioteca LIME de Symfony. Los desarrolladores correrán con la responsabilidad de la confección y realización de dichas pruebas a fin de garantizar la optimización del código de las aplicaciones.

Para el diseño de las interfaces web, se propone la utilización de Ext JS 3.3, originalmente concebida como una extensión de la Yahoo User Interface (YUI). Ext JS es una librería de Java Script (JS) con grandes prestaciones para el desarrollo de aplicaciones web y utiliza tecnologías como AJAX, XHTML/DHTML y DOM. También posee operatividad ligada al uso de JQuery y Prototype. Entre las ventajas que incentivan su uso se encuentran un alto grado de rendimiento dado el acabado en el código de JS con que cuenta. Pueden realizarse controles de usuario personalizables; cuenta con un modelo orientado a componentes correctamente diseñado y extensible. También posee una API de fácil utilización. Se distribuye haciendo uso de licencias tanto comerciales como Open Source.

Servidor web y de aplicaciones.

Partiendo del análisis previo realizado, se decidió establecer como servidor web y de aplicaciones el Apache 2.2.21, ya que tiene la capacidad de desempeñar ambas funciones. Entre las características de mayor peso en esta decisión se tiene su alta configurabilidad (existencia de numerosos parches y módulos, tanto para aumentar la seguridad y el rendimiento como para realizar integraciones con otras tecnologías). Otro motivo es el hecho de que producto de ser una tecnología libre, multiplataforma y con una gran comunidad de usuarios, se puede encontrar abundante documentación del mismo. Además, goza de un buen respaldo y actualización por parte de la fundación que le da soporte.

Sistema gestor de bases de datos.

Producto de su robustez y estabilidad se empleará como gestor de bases de datos PostgresSQL 9.0. Esta versión incorpora a sus funciones precedentes la “espera activa” y “replicación en flujo”, lo cual potencia aún más la escalabilidad en bases de datos de grandes prestaciones. Incluye mejoras en cuanto a administración y seguridad, específicamente en su sistema de actualización, verificación de fortaleza de contraseñas y autentificación. Permite el uso de disparadores (triggers) condicionales, por columnas y la aplicación de ordenamiento en funciones de agregación. Otras funcionalidades que han sido optimizadas son la mensajería de eventos (escucha/notificación), las consultas generadas por ORM (eliminación de JOIN) y planes de EXPLAIN para JSON Y XML. Como cliente se propone el uso de PgAdmin 1.12.0 o 1.12.3, que se distribuye junto a PostgresSQL y es una herramienta de gran aceptación y libre.

Políticas de trabajo en el entorno de desarrollo.

Para mantener la organización y disciplina necesarias en el entorno de desarrollo, se recomiendan emplear las siguientes pautas:

- ✓ Mantener actualizada toda la documentación del entorno de desarrollo.
- ✓ Realizar copias de seguridad de toda la documentación generada.
- ✓ Crear imágenes de los sistemas operativos y herramientas utilizadas en el desarrollo, documentando en cada caso las configuraciones empleadas.
- ✓ Actualizar periódicamente las imágenes utilizadas con versiones superiores estables del software utilizado o ante la necesidad de incluir algún otro que no se tuviese contemplado.
- ✓ Mantener actualizadas las estaciones de trabajo con la última imagen creada.

- ✓ Utilizar listas de verificación para comprobar el cumplimiento de estas actividades.

2.3.2 Área de despliegue.

El establecimiento de una plataforma de servicios constituye una tarea ardua en extremo e implica un alto grado de responsabilidad. Entre las actividades más significativas enmarcadas en esta área se encuentran: las pruebas a la arquitectura, la gestión de la monitorización y control de los activos de hardware que intervienen en el sostenimiento de la plataforma de servicios, el establecimiento de listas de verificación sobre los procesos que se realizan en este sentido y la confección de planes de prevención y estimaciones para hacer frente al aumento de la escalabilidad con el tiempo.

Pruebas de carga y estrés.

En el ámbito del despliegue, las pruebas de carga o estrés son aquellas que se realizan para determinar la escala en que un sistema es capaz de atender las solicitudes que le llegan. Para el desarrollo de estas pruebas los ingenieros se auxilian de herramientas informáticas que les permiten “simular” escenarios reales en los cuales evaluar el comportamiento de las aplicaciones. Es importante que los valores utilizados en dichos escenarios sean lo más exactos posibles, con el objetivo de que el margen de estimación obtenido tenga un alto grado de fiabilidad. Al llevarse a cabo las pruebas, han de variarse los parámetros de configuración de las mismas a fin de obtener el rendimiento que presenta el sistema ante situaciones de picos en la demanda, entre otros y así conocer hasta qué punto es capaz de responder tanto en ambientes controlados como excepcionales.

Esta clase de pruebas tienen gran importancia, ya que los resultados que arrojan sirven como medidores estadísticos para la asignación de los recursos necesarios para el despliegue. También aportan información de utilidad para los responsables del mantenimiento de los sistemas, contribuyendo a la detección en tiempo de anomalías que puedan dar al traste con el funcionamiento de las aplicaciones.

Jmeter 2.6.

Jmeter es una herramienta escrita en Java, con más de diez años de uso desde su liberación en 2001 por el Proyecto Jakarta que auspicia la fundación Apache. Originalmente fue concebida para el desarrollo de pruebas funcionales y de rendimiento a aplicaciones web. Hoy día cuenta con un mayor número de prestaciones para el trabajo con scripts Perl, bases de datos, servidores FTP, HTTP/HTTPS y más. Es capaz de realizar simulaciones utilizando muestras compuestas por un gran número de hilos y muestreo simultáneo de funciones por grupos de subprocesos separados. Permite llevar a cabo reproducciones de pruebas sin conexión mediante almacenamiento en caché de los resultados y cuenta con funcionalidades para la validación de los datos que recibe.

Requerimientos tecnológicos implícitos en el despliegue.

Antes de comenzar, se ofrecerán algunos conceptos necesarios para una mejor comprensión de las temáticas que se abordarán próximamente:

Virtualización: basa su funcionamiento en la utilización de máquinas virtuales. Comúnmente se aplica esta técnica en servidores dedicados con un hardware potente. Haciendo uso de herramientas como VMWare o Virtual Box, se fraccionan los recursos que posee el servidor asignándolos a las máquinas virtuales instaladas sobre este. El empleo de esta técnica resulta ventajoso ya que es posible utilizar de forma óptima los recursos disponibles, administrándolos o sustrayéndolos según las necesidades del sistema.

Clúster: se denomina clúster, en términos informáticos, a un conjunto de ordenadores conectados entre sí a través de una red de alta velocidad y que funcionan como un sistema integrado. Se emplean en el sostenimiento de servicios que requieren de un alto grado de disponibilidad, confiabilidad y eficiencia (dígase servidores web, bases de datos). Entre las ventajas que aporta su implantación, se tiene la posibilidad de balancear la carga de solicitudes realizadas al sistema entre los nodos que componen el clúster. De esta forma se obtiene un mejor desempeño y aprovechamiento de los recursos y aplicaciones.

En la definición de un escenario de despliegue intervienen múltiples factores. En materia de servidores, los elementos de mayor impacto sobre el rendimiento son: la memoria RAM, el procesador, velocidad de transferencia de la red, capacidad de almacenamiento, entre otros. Dadas las características que presentan las aplicaciones hacia las que va dirigida esta investigación, cabe destacar que estos parámetros están supuestos a cierta variabilidad, según la cantidad de recursos que demanden los posibles escenarios. La aplicación de técnicas de virtualización y una adecuada configuración de los servidores contribuirán significativamente a la optimización del manejo de los recursos disponibles; de ahí la recomendación de su inclusión como parte de las buenas prácticas a desarrollar en este proceso.

Se proponen como parámetros iniciales para la implantación de los sistemas las siguientes especificaciones. Los planes para el incremento de la escalabilidad de los sistemas en el tiempo también se describen a continuación:

Requerimientos del servidor web:

- ✓ RAM: 2 Gb

- ✓ Procesador: Intel Core 2 Duo a 3.0 GHz
- ✓ Capacidad del disco duro: 10Gb
- ✓ Red: 100 Mbps

Requerimientos del servidor de base de datos:

- ✓ RAM: 2 Gb
- ✓ Procesador: Intel Core 2 Duo a 3.0 GHz
- ✓ Capacidad del disco duro: 500Gb
- ✓ Red: 100 Mbps

Como base para la propuesta realizada se tienen las pruebas de estrés aplicadas al software ERP-Cuba (un software ERP - planificación de recursos empresariales- incluye módulos para la gestión logística, control de inventarios, producción y distribución de bienes, entre otros, de una empresa; por lo cual puede ser tomado como equivalente en robustez del tipo de aplicaciones a las cuales se centra esta investigación). En dichas pruebas se estableció una escala para medir el comportamiento de la aplicación de acuerdo al tiempo de respuesta del sistema / cantidad de usuarios concurrentes. Se obtuvo como resultado que la aplicación presentaba un rendimiento alto (tiempo de respuesta de 1 a 3 segundos, con RAM: 1Gb, procesador: Intel Pentium 4 a 3.0 GHz) para un número aproximado de 25 usuarios.

Otro factor tomado en consideración para la decisión, es un artificio matemático utilizado por los administradores de sistema para hacer una estimación del número máximo de clientes (atributo MaxClients en el caso de Apache) capaz de soportar el servidor web. La fórmula matemática es la siguiente:

MaxClients = Total RAM dedicada al servidor web (en Mb) / Tamaño máximo de los procesos hijos

Datos obtenidos en otras fuentes bibliográficas aseveran que en un servidor web que ofrece contenido dinámico, un proceso consume generalmente entre 1-30Mb de memoria. Por lo cual con los requerimientos iniciales propuestos, el sistema debe brindar servicios de forma estable a un número aproximado de 60 usuarios concurrentes (tomando como valor para la aproximación el pico máximo del intervalo: 30Mb, y reservando un espacio generoso de memoria para el SO, en este caso 203Mb).

Posteriormente, con el aumento progresivo del número de usuarios, es necesario definir un plan de escalabilidad que garantice mantener el rendimiento estable de las aplicaciones. En etapas tempranas de crecimiento es posible lograr esto incrementando variables del sistema como son la memoria RAM,

procesadores y espacio de almacenamiento. Una vez que el volumen de clientes alcance altos niveles, ante los cuales una estructura como la descrita con anterioridad no sea suficiente, se recomienda pasar a la implantación de clústeres para el manejo de los servidores web y de bases de datos que operan junto a estos. A partir de este instante el uso de virtualización, balance de carga y otras tecnologías tendrán un peso decisivo para el funcionamiento adecuado del software desplegado.

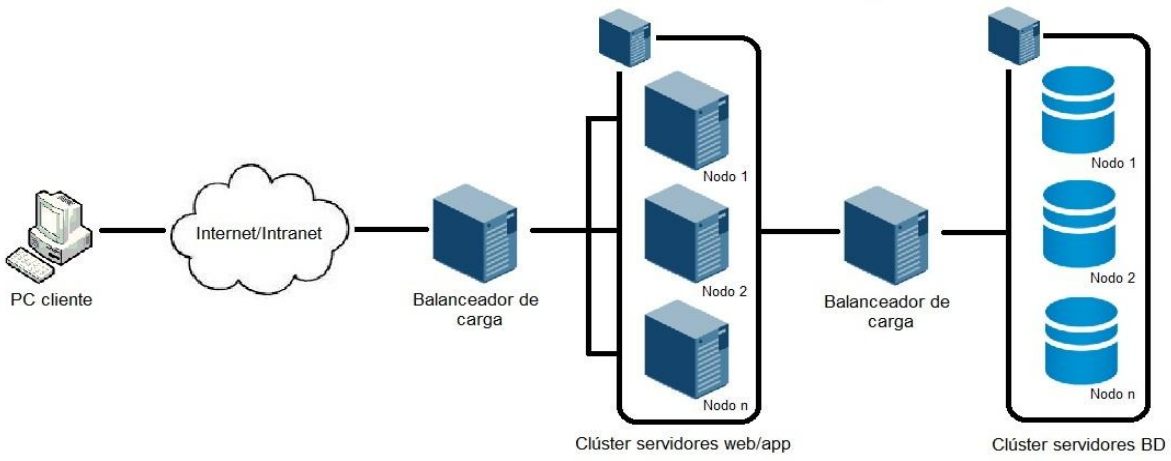


Figura 3: Ejemplo de una topología de red para brindar servicios de altas prestaciones (nótese el uso de tecnologías de clusterización y balanceo de carga).

Monitoreo y control.

Mantener un control estricto sobre los sistemas en explotación, las redes implicadas en ello y los recursos críticos que utilizan constituye una tarea de alta prioridad. Para llevar a cabo este trabajo, se utilizan herramientas de monitoreo estadísticas altamente especializadas. Entre las principales funcionalidades con que cuentan se hallan: generación de gráficos para la visualización de la carga de tráfico, comportamiento de los procesadores, consumo de RAM y ancho de banda, entre otros. De todos estos parámetros emiten reportes que permiten detectar las fallas o situaciones de riesgo que ocurren y de igual forma conocer cuáles son los servicios con mayor cantidad de solicitudes. Luego, con balanceadores de carga se distribuyen equitativamente las peticiones y de este modo se evita la sobrecarga de los servidores por la ocurrencia de cuellos de botella u otro tipo de interrupciones. A continuación se hará mención de un grupo de herramientas destinadas a estos fines, profundizando en sus características más notorias.

MRTG 2.17.4.

MRTG es una herramienta escrita en Perl y C, multiplataforma y distribuida bajo la licencia GNU. Creada para supervisar la carga de tráfico en la red, hoy día puede monitorear gráficamente (utilizando

el formato PNG) el estado de enrutadores, ordenadores y otros dispositivos conectados a esta. Cuenta con un alto nivel de configurabilidad, sin dejar de ser sencilla en este sentido. Producto de los lenguajes en que fue desarrollada, presenta un rendimiento excelente, dado que muchas rutinas críticas están escritas en C. El ser una herramienta que usa RRDtool le permite además almacenar datos estadísticos del tráfico monitorizado en las últimas, semanas, meses o años según la configuración inicial con que se cree la base de datos.

Cacti 0.8.7.

Cacti es una aplicación muy completa para la generación de gráficos de red escrita en PHP. Utiliza también herramientas RRDtool, contando con varios métodos para la recopilación de datos y manejo de usuarios. Permite hacer mediciones de temperatura, velocidad, consumo de energía, además de otras variables clásicas en este tipo de aplicaciones. Cuenta con una interfaz de usuarios amigable y es capaz de monitorear instalaciones de tipo LAN o redes complejas con un gran número de ordenadores.

Nagios 3.3.1.

Nagios es un software de monitorización de equipos y servicios de red muy popular, escrito en PHP y distribuido bajo licencia GPL. Desarrollado originalmente para ser ejecutado en GNU/Linux, cuenta con un desempeño adecuado en otros sistemas. Genera gráficas para visualizar el estado de las redes y recursos críticos, archivos de registros e historiales. Con Nagios puede realizarse monitorización de forma remota a través de túneles SSL cifrados o SSH. En caso de detectar anomalías, envía alertas de forma automática por correo electrónico o SMS a los encargados de las labores de vigilancia y control. Extensible, permite a los usuarios incorporarle nuevas funcionalidades según sus necesidades mediante plugins escritos en Bash, PHP, Python, C++ u otros lenguajes.

Webmin 1.580.

Webmin es una herramienta destinada a la configuración de sistemas Unix. Escrita en Perl y distribuida bajo licencia BSD. Accesible desde la web mediante una interfaz simple, permite administrar varios sistemas y servicios (servidores Apache, DNS, DHCP, MySQL, archivos de configuración, entre otros). Por defecto utiliza como protocolo de comunicación TCP a través del puerto 10000 y puede configurarse además para el soporte de SSL. Presenta una estructura modular, por lo que resulta fácil añadirle nuevas funcionalidades.

Bacula 5.2.6.

Está conformado por un conjunto de programas de código abierto que tienen como finalidad realizar el proceso de copias de seguridad (backups, verificación y restauración de datos) en redes heterogéneas de un entorno empresarial. Fácil de manejar y de rendimiento eficiente, dispone de funcionalidades avanzadas de automatización que normalmente requieren de la intervención de un administrador de sistemas. La mayor parte del código fuente de Bacula es distribuido mediante la licencia GPL 2.0. Entre los componentes que integra Bacula se encuentran:

Director: se encarga de gestionar todo lo relacionado a Bacula.

Console: es la aplicación que se utiliza para comunicarse con Director.

File: software encargado de realizar las copias de seguridad (debe estar instalado en la terminal).

Storage: se encarga de llevar a cabo los procesos de lectura y escritura en los medios de almacenamiento.

Catalog: gestiona el control de las bases de datos en uso.

Monitor: su función es llevar el historial del estado de las herramientas que componen Bacula.

Políticas de trabajo para el área de despliegue.

Al igual que en el entorno de desarrollo, en el despliegue es necesaria la inclusión de algunas de las buenas prácticas para asegurar su correcta implantación y soporte. Seguidamente se ponen a disposición algunas de las que se consideran de mayor prioridad:

- ✓ Documentar y guardar las configuraciones con que operan los servidores y sus bases de datos. De esta manera se minimiza el tiempo de afectaciones en el servicio producidas por fenómenos imprevistos. Es recomendable habilitar un sistema de salvas y réplica de las mismas auxiliándose para ello de herramientas y scripts que automaticen dichas funciones.

- ✓ Cualquier cambio a realizar en las configuraciones utilizadas, debe probarse primero en servidores de prueba ubicados en un ambiente controlado. Solo cuando se demuestre que las adaptaciones planificadas contribuyen al mejoramiento de los sistemas, serán aplicadas sobre los mismos.

- ✓ Actualizar permanentemente los registros de configuración y cambios de los servicios. Para esto es válido el apoyo en una bitácora en la que se reflejen debidamente estas actividades.

- ✓ Establecer listas de verificación que velen por el correcto uso y configuración de los servicios, aplicaciones y recursos claves que intervienen en su funcionamiento.

2.4 Vista de integración.

En la vista de integración se describe la forma en que se relacionarán los componentes, subsistemas y aplicaciones en los diferentes niveles en que se encuentran. Para ello se establecen los conectores a emplear por los diversos servicios que se definan, los estilos arquitectónicos hacia los cuales estará orientada la arquitectura, los patrones de diseño asociados a dichos estilos y el estándar de codificación sobre el que se va a llevar a cabo la implementación (para un estudio más profundo del tema, el lector puede remitirse al artefacto correspondiente a la Vista de integración).

2.4.1 Conectores.

Se entiende por conectores al grupo de tecnologías que son empleadas para la transmisión de información (mediante mensajes) entre los componentes, una vez que estos son identificados. Su formalización se establece mediante los contratos (que son los servicios o procedimientos) que requieren los componentes, implementados a su vez en interfaces que pueden encapsular varios servicios. Como tal, su función es la de establecer y normalizar el flujo de datos entre los componentes, pudiendo definirse para esto protocolos (principalmente SOAP) u otras variables de intercambio que den soporte al proceso. De inmediato se profundiza en un grupo de tecnologías con la aptitud necesaria para ser utilizadas con estos fines.

XML.

El Lenguaje de Marcación Extensible (XML, por sus siglas en inglés), surge producto del trabajo del W3C (World Wide Web Consortium) en 1998, teniendo como precedente al Lenguaje General de Marcas (SGML, por sus siglas en inglés). Es un lenguaje destinado al intercambio de datos entre diversas plataformas. Se encuentra compuesto por varias tecnologías, lo cual permite que pueda ser utilizado no solo en el entorno web, sino también en procesadores de texto, bases de datos y más; siendo precisamente su compatibilidad con otras tecnologías -que lo hace eficaz para la transmisión de información- su rasgo característico.

SOAP.

El Protocolo de Acceso Simple a Objetos (SOAP, por sus siglas en inglés) es un protocolo para la transmisión de datos entre aplicaciones web. Un mensaje SOAP no es más que un documento XML conformado por un “sobre” que posee un mecanismo de codificación estándar para el cifrado de la

información que contiene. Este sobre a su vez, contiene los datos a transmitir en sí y cuenta además con una cabecera con información referente al mensaje en cuestión. Típicamente se emplea una estructura compuesta por dos mensajes SOAP, uno para enviar la solicitud y el otro para la respuesta, a fin de facilitar el proceso de comunicación.

JSON.

La Notación de Objetos de Java Script, conocido popularmente como JSON, es una notación sencilla para el intercambio de datos. Dada su simplicidad, es empleado como alternativa al XML, aunque perfectamente pueden ser utilizados en combinación. Cuenta con una estructura universal (una colección de pares nombre/valor y una lista ordenada de valores), que es reconocida virtualmente por todos los lenguajes de programación. Goza de fácil interpretación y generación por parte de los ordenadores.

AJAX.

Es una tecnología que corre en la vista, del lado del cliente. AJAX (Java Script Asíncrono y XML) resulta de la agrupación de varias tecnologías (XHTML, CSS, XML, JSON, DOM, entre otras) unificadas a través de Java Script. Tradicionalmente, al usuario realizar una petición mediante su navegador web, esta es enviada al servidor que devuelve una nueva página en HTML para ser cargada por el navegador. Ahora bien, al introducirse AJAX en este modelo, la petición realizada por el usuario es antes analizada por AJAX, que la procesa. Posteriormente, envía una nueva solicitud al servidor con las especificaciones dictadas por el usuario. De esta forma, el servidor web en lugar de devolver toda una nueva página (siempre que no sea necesario) solo envía los datos de esta que han solicitado ser actualizados por AJAX, que se encarga luego de "refrescarlos" en el navegador. En la Figura 4 (Eguíluz Pérez, 2008) se evidencia de forma transparente esta situación.

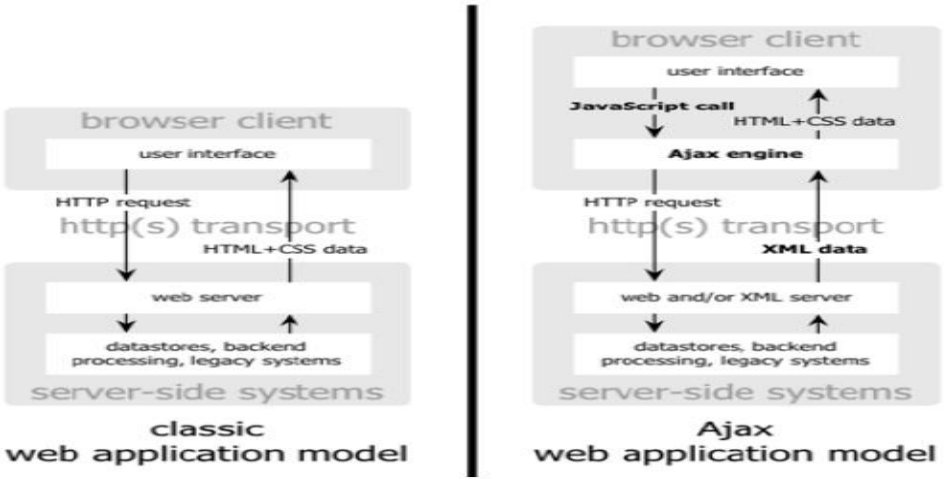


Figura 4: Diferenciación entre una solicitud web tradicional y el esquema implementado por AJAX.

Con la conjugación de los elementos mencionados con anterioridad y la implementación de otros como el patrón y estilo de Inversión de Control (documentado en la sección correspondiente a los patrones), se puede llevar a cabo una estandarización de la comunicación tanto en el nivel interior como exterior de los sistemas. Sin embargo, la definición y establecimiento de los conectores es solo la fase inicial del proceso de integración. Por sí solos estos no tienen la capacidad de hacer funcionar como un todo el complicado entorno que resultan los sistemas descritos. Se impone por esto la necesidad de vincular dentro de la arquitectura de integración los patrones y estilos arquitectónicos que logren consolidarla desde el punto de vista estructural. En el siguiente apartado se dan muestras de la relación existente entre conectores y los mencionados estilos.

2.4.2 Estilos arquitectónicos.

Los estilos arquitectónicos determinan en buena medida el sentido en el cual guiar el desarrollo del software y por tanto, los resultados obtenidos de dicho proceso. Dadas las características del modelo de negocio y aplicaciones descritas en este trabajo, existen un conjunto de estilos que identifican la arquitectura propuesta. Unos lo hacen de forma más representativa mientras otros se desenvuelven como complementos de la arquitectura en un segundo plano. A continuación se hará una presentación de los estilos seleccionados en orden de prioridad.

SOA.

Resulta el estilo arquitectónico que por excelencia identifica la arquitectura propuesta (Figura 5, Alba, 2008). El término de servicios no es tratado como una generalidad, sino que va más bien dirigido a los servicios web basados en XML y utilizando el protocolo SOAP. Un servicio web por su parte, es visto como una entidad de software que encapsula funcionalidades de la lógica del negocio y que puede servir dichas funcionalidades a otras entidades a través de interfaces bien definidas (descritas en un formato procesable por máquina, específicamente WSDL). Los servicios, se encuentran acoplados a un nivel mínimo, contando con la posibilidad de recibir peticiones de cualquier procedencia; a su vez los componentes que requieran de un servicio pueden descubrirlo y consumirlo de forma dinámica haciendo uso de un componente que actúe como mediador o mediante la implementación del patrón de Inversión de Control (IoC, por sus siglas en inglés).

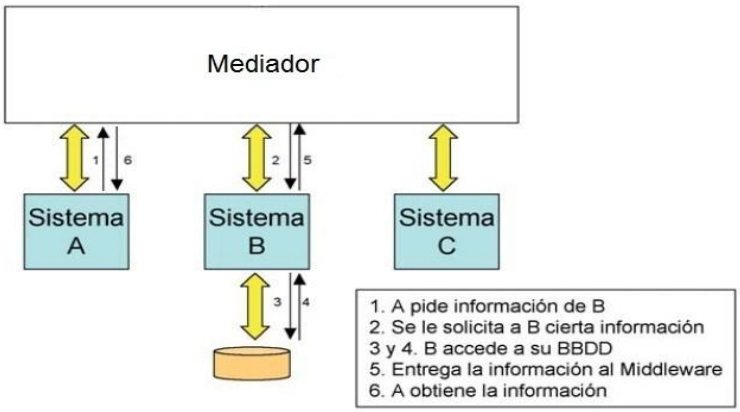


Figura 5: Sistemas relacionados entre sí mediante el estilo SOA.

MVC.

Modelo – Vista – Controlador es el estilo arquitectónico utilizado por Symfony, separando la vista de las aplicaciones de la lógica del negocio y del controlador. Para el tipo de aplicaciones descrito, la vista quedaría conformada haciendo uso de Java Script (Ext JS) del lado del cliente y comunicándose con los componentes del lado del servidor (Controlador y Modelo) mediante el uso de AJAX. Las respuestas a estas solicitudes serán devueltas en XML o JSON según corresponda a la petición realizada.

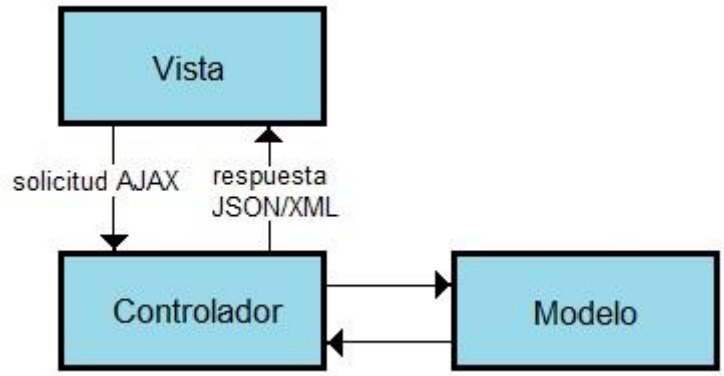


Figura 6: Representación gráfica del estilo arquitectónico MVC.

Arquitectura en capas.

Consiste en organizar los componentes del sistema en capas de forma jerárquica. Cada capa tiene asignadas un grupo de responsabilidades y consume servicios de sus capas inferiores. Comúnmente se identifican capas como la de presentación, negocio, capa de acceso a datos, datos e infraestructura. Pueden incluirse otras según las necesidades de adaptación del sistema, mas ha de hacerse de manera consensuada ya que esto tiende a elevar la complejidad del mismo.

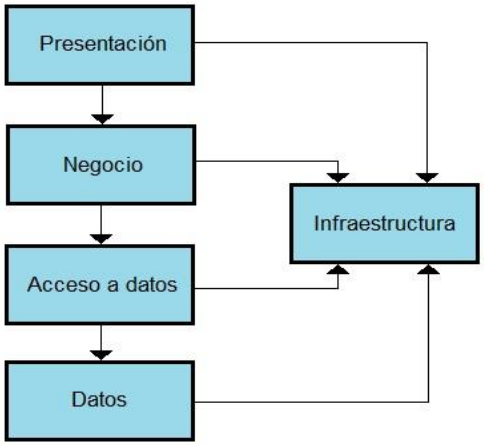


Figura 7: Arquitectura en capas.

Cliente – Servidor.

Constituye una especialización del estilo en capas. Se presenta de forma independiente a este para un mejor entendimiento por parte del lector. Consiste en el alojamiento de los servicios (aplicaciones) en uno o más nodos servidores (web, de aplicaciones y bases de datos) a los cuales acceden los usuarios mediante un navegador para consumirlos.

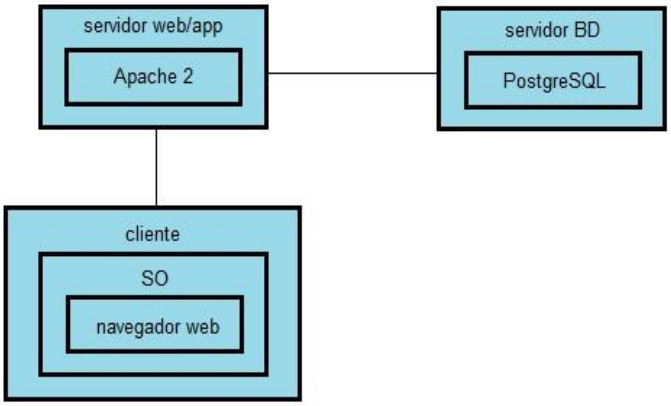


Figura 8: Estilo Cliente - Servidor.

Tuberías y filtros.

Constituye un estilo de gran simplicidad, representado en un buen número de arquitecturas. Se emplea en situaciones típicas donde el usuario introduce peticiones y/o datos al sistema, que procesa dichas entradas y las transforma gradualmente en salidas que son devueltas al usuario en un formato estándar.

Figura 9: Implementación del estilo Tuberías y filtros en un compilador.

2.4.3 Patrones.

Partiendo de los argumentos dados en el capítulo anterior con relación al uso de los patrones, se presentan en este epígrafe los que son considerados de necesaria inclusión en la arquitectura propuesta. En cada caso se hará una descripción de los mismos a fin de reflejar los motivos por los que han sido seleccionados.

Patrón y estilo de Inversión de Control (IoC).

La utilización de este patrón se refleja en la implementación de un fichero XML, que actúa como mediador para la integración de subsistemas. En este fichero, cada subsistema publica los servicios con que cuenta, ocurriendo de igual forma en el interior de cada uno, donde también se crea un fichero (IoC interno) que contiene los servicios que cada componente brinda al resto de los que integran el subsistema. Por su parte, cada componente cuenta con una interfaz en la cual se definen los servicios que posee. Para establecer la comunicación con otras aplicaciones externas, se implementa además una interfaz donde se ubican todos los servicios brindados por el sistema. Para facilitar un mejor entendimiento de lo anteriormente explicado se muestra la situación expuesta gráficamente en la Figura 10.

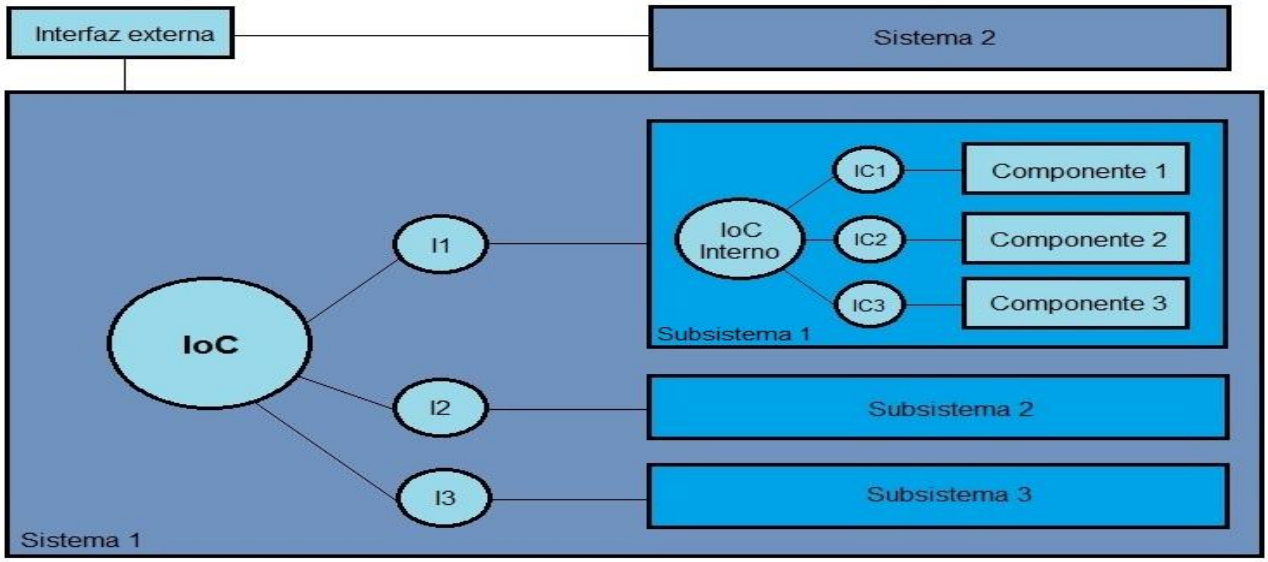


Figura 10: Representación gráfica de la implementación del patrón IoC.

Patrones GRASP.

Los Patrones de Software para la Asignación General de Responsabilidades (GRASP, por sus siglas en inglés) se dividen en dos grupos. En el grupo principal se encuentran los patrones Experto, Creador, Controlador, Alta Cohesión, Bajo Acoplamiento; y entre los pertenecientes al grupo de apoyo se tienen los patrones: Indirección, Fabricación Pura, Polimorfismo y Variaciones Protegidas. A continuación se profundiza en aquellos que se considera servirán de apoyo para el desarrollo de las aplicaciones.

Experto en información: este patrón desempeña el principio básico de asignación de responsabilidades. Plantea que la implementación de una funcionalidad o la creación de un objeto determinado deben llevarse a cabo por la clase que contiene toda la información necesaria para ello. De esta forma se garantiza una alta cohesión en el diseño y por tanto disminuye el nivel de acoplamiento en la aplicación.

Creador: el patrón creador tiene la responsabilidad de definir la clase encargada de la instanciación o creación de un nuevo objeto o clase. Las nuevas instancias deben ser creadas por aquella clase que reúna la información necesaria para ello, haga uso directo de las instancias creadas, las contenga o maneje por sí misma.

Controlador: es un patrón que funciona como intermediario entre una interfaz y el método que la implementa. Recibe los datos introducidos por el usuario y los distribuye a las clases encargadas de procesarlos según el método especificado. Como tal, separa la lógica de la aplicación de la capa de presentación, aportando un mayor grado control y potencia la reutilización. La utilización de varios controladores contribuye al aumento de la cohesión a la vez que disminuye el acoplamiento.

Alta cohesión: los conceptos de cohesión y acoplamiento están estrechamente relacionados y son inversamente proporcionales. En un sistema donde sus componentes (clases) se encuentran altamente cohesionados (la información que contienen corresponde en buena medida al papel que desempeña dentro de la aplicación) resulta en un menor grado de dependencia y por tanto del acoplamiento que pueda existir entre los mismos.

Bajo acoplamiento: el patrón de bajo acoplamiento establece la necesidad de que el nivel de dependencia entre las clases sea el menor posible. De esta forma al modificarse algún elemento, la repercusión del cambio sobre el resto es mínima, potenciando la reutilización de los componentes dentro del sistema.

Fabricación pura: su utilización viene dada cuando surge la necesidad de incluir clases que inicialmente no se concibieron para la solución del problema. Esta inclusión se produce al encontrarse una clase con un pobre nivel de cohesión, sin que exista otra que pueda contener las funcionalidades necesarias. Por tanto, el diseñador agrega una nueva clase con la cual logra aumentar la cohesión y reutilización en el sistema y mejorarlo desde el punto de vista estructural. Se destaca sin embargo, que el uso abusivo de este patrón puede incrementar la complejidad de la aplicación al incluirse clases que pueden contener una sola funcionalidad.

Variaciones protegidas: constituye una defensa frente a cambios en el futuro. Ante la posibilidad de variaciones de los elementos no concebidas inicialmente, estos son envueltos en interfaces de las cuales se realizan varias implementaciones (a través del polimorfismo). Por esta vía, se previenen o disminuyen afectaciones al sistema producidas por un cambio en el futuro de algún componente.

Patrones asociados al uso de Symfony.

Observador (Observer): es un patrón de comportamiento. Se utiliza para mantener “informados” a objetos desacoplados entre sí y que presentan dependencia de un objeto, notificando a los primeros ante cualquier cambio que ocurra en el último. Este patrón es utilizado tradicionalmente en la capa de presentación en el diseño de componentes para las interfaces de usuario de las aplicaciones.

Controlador frontal: se comporta como un controlador dentro de la capa de negocio, que concentra todas las peticiones hechas a la aplicación para luego asignarlas a manejadores específicos, usualmente comandos.

Instancia única (Singleton): conocido también como patrón Singleton, opera en el nivel de la capa de negocios. Se utiliza en clases que se necesitan tengan una sola instancia (objeto), encargándose además de proveer un punto de acceso global a la misma.

Orden (Command): empleado también en la capa de negocio, su función consiste en el encapsulamiento de acciones que den cumplimiento a una petición en un objeto. En el caso particular del framework Symfony, las acciones son evidencia del uso de este patrón.

Filtros interceptores: implementado por Symfony en la capa de negocios, de gran utilidad en el manejo de validaciones, conversiones y otras cuestiones relativas al tema de la seguridad. Su función está asociada a la realización de procesos antes y después de la ejecución de una acción.

Registro activo: se concibe con el objetivo de aportar un objeto homólogo a una tupla de una entidad en la base de datos, encapsulando su acceso y la lógica de manipulación de los datos (inserción, actualización, carga o eliminación). Este patrón es implementado por Propel en la capa de acceso a datos.

Mapeo de metadatos: utilizado por Propel en la capa de acceso a datos, teniendo como motivo de existencia el registro de la información del mapeo de objetos relacional realizados.

Objeto de consulta: su función en la capa de acceso a datos consiste en la encapsulación de las consultas realizadas a la base de datos.

2.4.4 Estándar de codificación.

Definir un estándar de codificación en el ambiente de integración resulta de vital importancia. La facilidad de lectura, comprensión, mantenimiento y depuración son algunas de las características que deben presentar para ser considerados aptos. Dichos estándares deben garantizar además que se escriba un código eficiente, óptimo, que consuma el mínimo de recursos (tiempo y memoria). Se recomienda en esta situación la adopción de los Estándares de codificación para PHP v1 (contenido en el expediente de proyecto propuesto por la guía base metodológica para el diseño de arquitecturas), siendo posible su modificación por parte de los arquitectos principales del departamento en algunos aspectos que propicien su adaptación a la experiencia y modus operandi de los equipos de desarrollo que lideran.

2.5 Vista de despliegue.

En la vista de despliegue se realiza un análisis de varios puntos que inciden sobre la forma en que se ha de realizar el despliegue y comercialización de las aplicaciones. Entre estos factores se tienen el tipo de licenciamiento a emplear, modelos de negocio, requerimientos mínimos para realizar el despliegue y otras actividades asociadas al mismo. En relación a los modelos de negocio, es recomendado el utilizar modelos híbridos que tomen lo mejor del sector de código abierto y del privativo. Esta vía garantiza la disminución de los costos de producción, un mayor aseguramiento de la calidad y facilidades para el licenciamiento comercial. En la guía metodológica se establecen una serie de tópicos en los cuales desglosar la vista de despliegue para facilitar su elaboración, los cuales serán abordados a continuación (para un estudio más profundo del tema, el lector puede remitirse al artefacto correspondiente a la Vista de despliegue).

2.5.1 Caracterización de la solución vista desde las licencias de los componentes que la forman.

Por licencia de software se conoce al contrato establecido entre el desarrollador/autor de un determinado software sobre el cual ejerce derechos de autor y propiedad intelectual y el usuario/consumidor del mismo. En dicho documento legal se definen estrictamente los derechos y condiciones bajo las cuales el usuario hará uso del software (modificaciones, distribución, período de validez de la licencia y muchas otras) y las posibles obligaciones contraídas por el autor/licenciante en virtud del acuerdo tomado.

Atendiendo al grado de libertad o concesión de derechos que otorgan cada una de las licencias, estas se clasifican en cuatro grupos fundamentales:

Licencias de código abierto permisivas (copyleft débil): estipulan la creación de obras derivadas sin que estas tengan obligaciones de protección. Algunas de las licencias pertenecientes a este grupo y presentes en las herramientas propuestas para el desarrollo en este trabajo son:

- ✓ Licencia MIT: contemplada para la distribución de Symfony.
- ✓ Licencia BSD: utilizada para el licenciamiento de PostgreSQL.
- ✓ Licencia PHP: tiene un estilo similar al de la BSD, da soporte al uso del lenguaje PHP5.
- ✓ Licencia Apache 2.0: empleada para regir la distribución y uso del servidor web y de aplicaciones Apache.

Licencias de software de código abierto robustas o de copyleft fuerte: en su contenido existen cláusulas que indican que las obras derivadas o modificaciones realizadas al software original deben ser licenciados bajo los mismos términos expresados en la licencia original. Como ejemplo de licencia correspondiente a este grupo se encuentra:

- ✓ Licencia GPL: se utiliza en combinación con la CDDL (que no pertenece a esta clasificación) para la distribución del IDE NetBeans.

Licencias de código abierto robustas con copyleft débil o híbridas: dedican una o más cláusulas a imponer la obligación de que todas las obras resultado de la modificación de un original se deban licenciar con la misma licencia de este; mas las obras derivadas que se puedan obtener de su uso pueden ser licenciadas bajo términos o condiciones diferentes. Ejemplo:

- ✓ Licencia CDDL (Licencia de Distribución para el Desarrollo Común): es uno de los componentes presentes en la licencia dual del NetBeans.

Licencias propietarias: como su nombre lo indica, a este grupo pertenecen las licencias que han de ser compradas para poder hacer uso de un determinado producto, o bien la licencia creada por la propia entidad para la protección de los bienes que comercializa. En el software empleado que tiene este tipo de licencia se hallan:

- ✓ Ext JS y Visual Paradigm (en el caso de la herramienta de modelado tiene un licenciamiento dual, presenta una licencia académica –libre, solo para fines docentes- y otra para el desarrollo de productos con fines comerciales, a la cual se hace referencia en este caso).

En el caso particular del modelo de negocio tratado en el presente estudio, es necesario recordar que por el modo en que este establece los contratos con los usuarios, no se estipula un licenciamiento directo de las aplicaciones. Este factor está determinado por el hecho de que el código fuente no será entregado a entidades externas, que solo pagarán por el uso/consumo que hagan del sistema. Sin embargo, es importante realizar un estudio relativo al tema del licenciamiento del software por la repercusión que tiene este en el campo legal, de ahí la exposición de los conceptos, criterios y datos presentados con anterioridad.

2.5.2 Caracterización de la solución vista desde las estrategias de desarrollo empleadas para su elaboración.

En concordancia las licencias empleadas por los componentes seleccionados para el desarrollo de las aplicaciones, abordadas en el primer punto de esta vista, se decide optar por la estrategia del desarrollo híbrido. Al hablar de este se hace referencia a modelos en que el software se distribuye mediante licencias de código abierto, pero con funcionalidades que han sido creadas a puertas cerradas; en este campo se identifican tres categorías distintivas:

Proveedores híbridos: el desarrollo es llevado a cabo por miembros de una sola organización, mas algunas de las funcionalidades han sido implementadas a puertas cerradas.

Comunidad híbrida: la mayor parte del código es generado por comunidades públicas, añadiéndose funcionalidades desarrolladas a puertas cerradas por las entidades proveedoras del producto o servicio.

Híbrido mixto: el producto o servicio combina elementos creados a partir de proyectos de código abierto desarrollados por comunidades u otros proveedores y además se le adicionan funcionalidades implementadas dentro de la propia organización a puertas cerradas.

Atendiendo a las clasificaciones anteriores, se decide en favor de la estrategia de desarrollo híbrida en su variante mixta por ser esta la que se ajusta con mayor claridad a las características de los procesos y soluciones desarrollados por el departamento.

2.5.3 Definición del tipo de licenciamiento de la solución vista desde la arquitectura.

Tomando como punto de partida los análisis previos, se observa el uso conjugado de licencias de código abierto permisivas, código abierto no permisivas y licencias privativas, que intervienen en la composición de la solución. Esta estrategia va dirigida a aprovechar las ventajas que se desprenden del uso de cada uno de estos modelos de licenciamiento, y responde desde el punto de vista de la arquitectura a un Licenciamiento Abierto y Cerrado. Este tipo de licenciamiento precisa que los proyectos de código abierto pueden ser desarrollados a través de la combinación de componentes independientes que luego pueden distribuirse mediante licenciamiento comercial como código cerrado.

2.5.4 Definición de las estrategias de ingreso vistas desde la arquitectura.

Dado el modelo de negocio planteado para el desarrollo de la investigación, la arquitectura de las aplicaciones se ha moldeado con el objetivo de garantizar que la principal estrategia de negocio para la obtención de ingresos resulte del Software como Servicio (SaaS). El funcionamiento de este modelo, ampliamente documentado en el capítulo introductorio, está dado por el hecho de que el usuario paga por el acceso y consumo de las aplicaciones vía internet. Como complemento o paralelamente pueden utilizarse además métodos basados en suscripciones (generalmente se establecen por un período de tiempo determinado) en los cuales se pueden definir contratos de soporte, atención personalizada u otras actividades que puedan aportar ganancias adicionales a la organización.

2.5.5 Estrategias de empaquetamiento para la implantación, instalación, soporte y actualización de las soluciones.

Producto de las características del propio SaaS, no es necesaria la confección de un kit de instalación para la implantación externa de las soluciones. Las mismas se mantienen de forma centralizada en una plataforma atendida directamente por el proveedor de servicios (Tabla 5) sobre el cual recae esta responsabilidad. A manos del cliente llegan únicamente los manuales de usuario o ayudas online

entregadas en formato digital (Tabla 6) una vez que este contrae un contrato para el consumo de uno o varios de los servicios ofertados.

Tabla 5: Requerimientos mínimos para la instalación de la solución (transparente para el usuario final, dado el modelo de negocio adoptado constituye una responsabilidad del proveedor).

Hardware (Servidor web)	
Cantidad	1
CPU	Intel Core 2 Duo a 3.0 GHz
RAM	2 Gb
HDD	10 Gb
LAN	2 x NIC, 1 Gbit
Fuente de Alimentación	1 x 800W
Hardware (Servidor de BD)	
Cantidad	1
CPU	Intel Core 2 Duo a 3.0 GHz
RAM	2 Gb
HDD	500 Gb
LAN	NIC 1 Gbit
Fuente de Alimentación	1 x 800W

Tabla 6: Ejemplo de artefactos o código ejecutable que pueden ser entregados al usuario como medios de soporte para el manejo de aplicaciones de tipo SaaS.

Nombre	Descripción	Formato en que se entrega (PDF, Ejecutable)	Soporte en que se entrega (CD, Online, DVD, Instalado)
Manual de usuario GDR.	Sistema para la confección dinámica de reportes logísticos para la gestión empresarial.	PDF	Online

Queda por parte de los arquitectos y líderes de proyectos, el definir los niveles de soporte que serán capaces de ofrecer para cada una de sus aplicaciones y el período durante el que darán el servicio. También han de fijar un margen de tiempo para las actualizaciones del software. Este debe ser factible para los clientes y a la vez debe permitir al equipo de desarrollo llevar a cabo la corrección de bugs y la implementación de nuevas funcionalidades de utilidad para los usuarios que eleven la calidad y prestaciones del sistema. Se destaca como aspecto positivo del proceso de actualización que este ocurre de modo transparente a la locación del usuario ya que toda la lógica e infraestructura de soporte de las aplicaciones se encuentra en manos del proveedor.

2.6 Conclusiones del capítulo.

Durante el desarrollo de este capítulo, fueron elaboradas las propuestas de las vistas de la arquitectura más representativas con relación al modelo de negocio SaaS. A modo de resumen y con basamento en la guía base metodológica para el diseño de Arquitecturas, fueron expuestas temáticas asociadas a:

- ✓ Las principales herramientas y tecnologías (software y hardware) a utilizar para el desarrollo, mantenimiento, monitorización y explotación de las aplicaciones.
- ✓ Las configuraciones de software, estándares y políticas de seguridad a implementar para garantizar la organización y disciplina durante todo el proceso, que tributen al aumento de la calidad final del producto.
- ✓ Los patrones, estilos y restantes elementos de la arquitectura con mayor impacto sobre sistemas orientados a servicios y sus relaciones desde puntos de vista estructurales y de integración.
- ✓ Cuestiones vinculadas al despliegue de las aplicaciones: licenciamiento, estrategias de comercialización, implantación, soporte, entre otras.

Capítulo 3: Evaluación de la Arquitectura Propuesta.

Las vistas arquitectónicas desarrolladas en el capitulo anterior, conforman una arquitectura “genérica” que responde a la concepción de una línea de productos orientada al modelo comercial SaaS. La misma puede ser empleada como referencia para el desarrollo de aplicaciones orientadas a servicios, complementando la arquitectura particular de cada producto que se desarrolle con estos fines. Para realizar las mediciones a esta arquitectura propuesta se hará uso del método de evaluación ATAM, teniendo como atributos de calidad a medir los establecidos por Losavio (14) en su adaptación del estándar ISO/EC 9126 (dicho estándar tiene como meta identificar aquellos atributos de calidad con los que debe cumplir todo software). Estos son: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad. Como consideración del autor, además, se estipula la inclusión del atributo *Escalabilidad*, dada la importancia que deriva su satisfacción para el tipo de aplicaciones perfiladas a lo largo del trabajo.

3.1 Evaluación de la Arquitectura mediante ATAM.

El método de evaluación ATAM se encuentra dividido en cuatro fases que comprenden nueve pasos en total, que se describen a continuación:

Tabla 7: Fases y pasos del método de evaluación ATAM.

Fase 1: Presentación.	
Presentación del ATAM	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.
Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.
Presentación de la arquitectura	El arquitecto describe la arquitectura, enfocándose en cómo esta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis.	
Identificación de los enfoques arquitectónicos	Estos elementos son detectados, pero no analizados.
Generación del Árbol de Utilidad	Se establecen los atributos de calidad que engloban la “utilidad” del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, entre otros), especificados en forma de escenarios. Se anotan los estímulos, respuestas y se establece la prioridad entre escenarios.
Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos

	del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3: Pruebas.	
Lluvia de ideas y establecimiento de prioridad de escenarios	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte.	
Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Como resultado de la aplicación del método ATAM para la evaluación de la arquitectura, al construirse el Árbol de Utilidad (5to paso) se refinaron los atributos de calidad seleccionados, quedando estructurado de la siguiente forma:

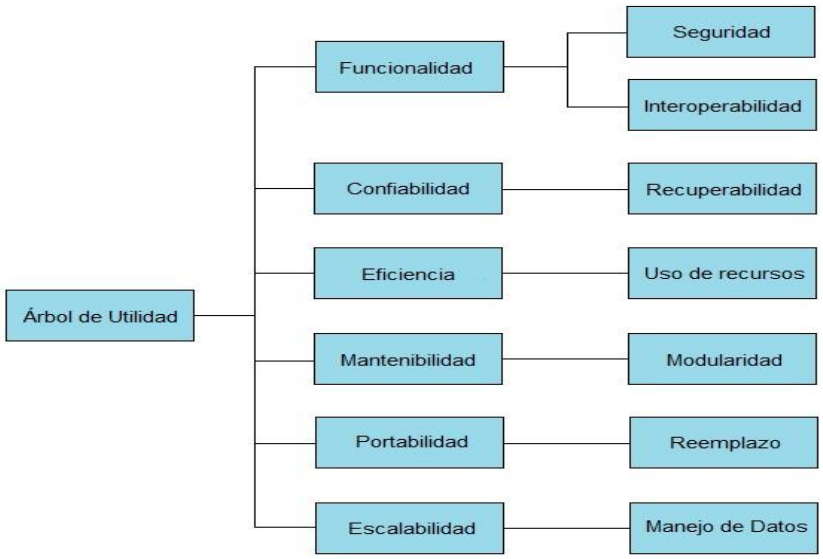


Figura 11: Árbol de Utilidad de la Arquitectura.

Una vez conformado el Árbol de Utilidad del sistema, tuvo lugar una tormenta de ideas que permitió identificar los escenarios utilizados como bases para probar la arquitectura propuesta. Durante el desarrollo de esta actividad, además, se estableció un orden de prioridad para cada escenario planteado, atendiendo al grado de relevancia del atributo de calidad a medir en cada situación y el impacto que deriva su materialización en la arquitectura.

3.1.1 Análisis de los principales escenarios identificados.

Del análisis detallado de los escenarios considerados de mayor prioridad para la arquitectura candidata, se obtuvieron como resultados los listados correspondientes a las relaciones identificadas entre los atributos, puntos sensibles, así como los riesgos y no riesgos que caracterizan a la arquitectura propuesta. A continuación se ofrece una muestra de los escenarios diseñados (los escenarios restantes se encuentran anexados al final del presente documento), para cada uno de ellos se describe la motivación que le da causa, el ambiente concebido para el sistema y la respuesta que se obtiene al introducirse el estímulo. En cada caso se ofrece también un argumento dado desde el punto de vista arquitectónico de las decisiones tomadas.

Tabla 8: Análisis del escenario 3.

Escenario #3:	**Descripción:**	Un usuario que se encuentra operando con un subsistema/sistema específico, desea hacer uso de una funcionalidad perteneciente a otro subsistema/sistema externo asociado al que opera actualmente, sin necesidad de invocar al programa en su totalidad. Al realizar la solicitud, el sistema la recepciona, identifica el componente externo que contiene la funcionalidad, procesa la solicitud y devuelve el resultado esperado al usuario.
Atributo(s):	Funcionalidad - Interoperabilidad.	
Ambiente:	Trabajo Normal del Sistema.	
Estímulo:	Se intenta consumir un recurso perteneciente a un sistema externo a la aplicación que guarda relación con la misma.	
Respuesta:	Alternativas: Si el usuario posee los permisos suficientes para ejecutar la presente solicitud: el sistema invoca a la interfaz del servicio solicitado transfiriendo la petición, el servicio la procesa y devuelve el resultado al sistema para que lo presente al usuario. Si el usuario no cuenta con los permisos para invocar al servicio solicitado: el sistema informa al usuario de la negación del servicio y expone las causas del hecho.	
Decisiones arquitectónicas	**Razonamiento**	
Arquitectura Orientada a Servicios y Patrón de Inversión de Control.	La arquitectura SOA se enfoca al trabajo con servicios web, basados en XML fundamentalmente, utilizando SOAP como protocolo de comunicación. Cada servicio posee una interfaz en la cual publica la lógica del negocio que encapsula (funcionalidades del sistema) y se vale de esta para informar al fichero IOC de su estado. Cuando uno es invocado, el IOC se encarga entonces de localizar el servicio solicitado, propiciando un bajo nivel de acoplamiento. A raíz de establecer la comunicación de una forma más eficiente, se plantea el uso de IOC para los servicios pertenecientes a un mismo sistema, dejando a	

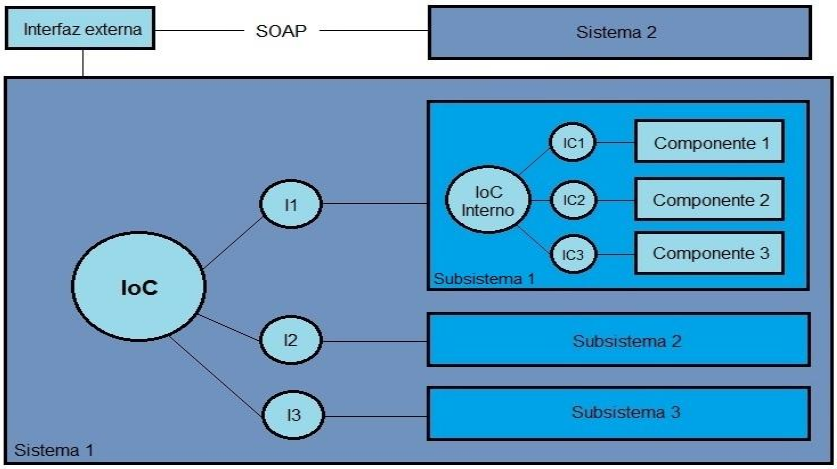
	cargo de SOAP la interacción con sistemas externos.
Representación gráfica.	

Tabla 9: Análisis del escenario 6.

Escenario #6:	**Descripción:**	El usuario trabaja con la aplicación en un entorno de bajas prestaciones (cliente ligero). El sistema ejerce una demanda mínima de recursos para el hardware del cliente dado que el grueso de la aplicación se corre del lado del servidor.
Atributo(s):	Eficiencia - Uso de recursos.	
Ambiente:	Trabajo Normal del Sistema.	
Estímulo:	El sistema recibe la solicitud con los datos introducidos y/o solicitados por el usuario.	
Respuesta:	El sistema procesa la petición procedente de la vista del lado del cliente utilizando la capacidad de cómputo ubicada del lado del servidor. Inmediatamente retorna los datos solicitados al navegador web del lado del cliente.	
Decisiones arquitectónicas	**Razonamiento**	
Estilos arquitectónicos M-V-C y Cliente - Servidor.	El uso combinado de los estilos arquitectónicos M-V-C y Cliente - Servidor permite separar la lógica de la aplicación (del lado del servidor) de la vista con que interactúa el usuario (ubicada del lado del cliente). El hardware del usuario se libera de invertir una mayor cantidad de recursos en el sostenimiento total del sistema (díganse ciclos de procesador, RAM, espacio de almacenamiento). Su responsabilidad para con la aplicación recae en el manejo de las tecnologías que intervienen en el proceso de comunicación y transmisión de datos (HTTP/HTTPS, JSON/XML, JavaScript, HTML) que no demandan un gasto excesivo de recursos.	

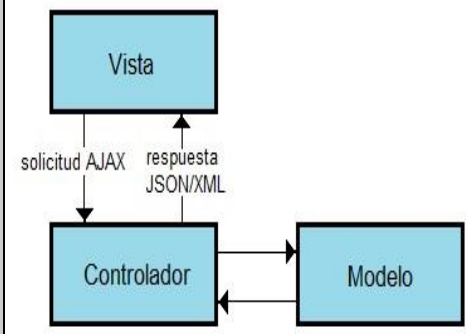
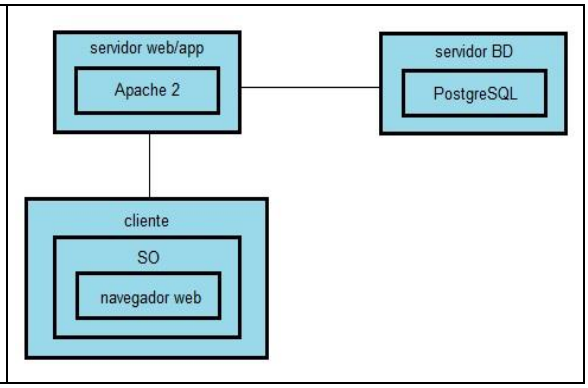
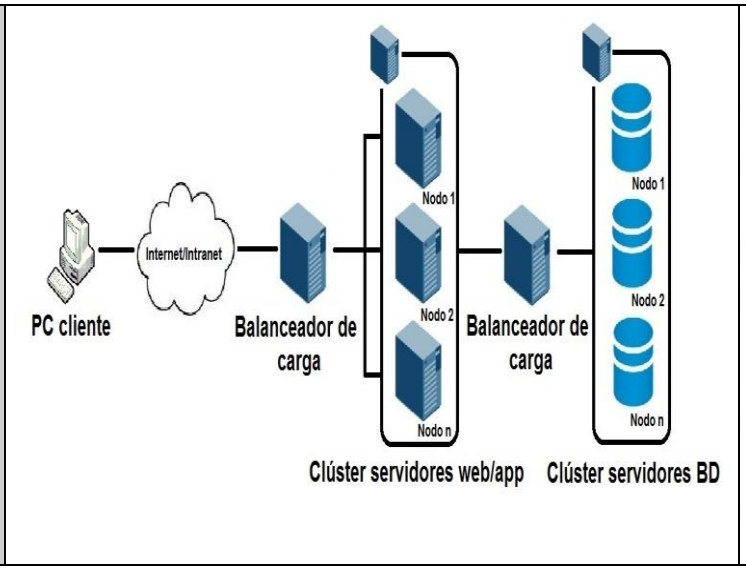
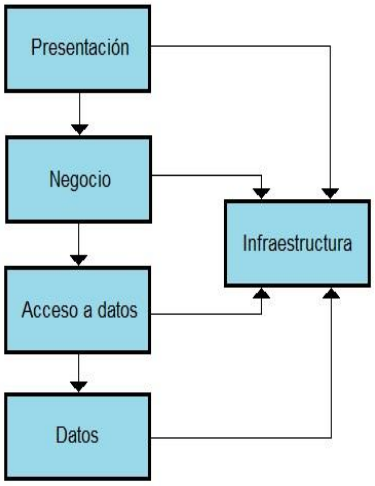
Representación gráfica.		

Tabla 10: Análisis del escenario 7.

Escenario #7:	**Descripción:**	El sistema debe ejecutar en paralelo múltiples procesos realizados por los usuarios. Debe cumplir con el nivel de desempeño deseado, manejando adecuadamente el incremento en los volúmenes de procesamiento producidos por el aumento del número de transacciones o cantidad de usuarios concurrentes operando sobre el mismo, sin que se afecte en una medida significativa su funcionamiento.
Atributo(s):	Escalabilidad - Manejo de datos.	
Ambiente:	Aumento sistemático del consumo de los recursos con que cuenta el sistema para su funcionamiento.	
Estímulo:	La cantidad de clientes atendidos por el sistema se incrementa, provocando un uso intensivo del hardware asignado.	
Respuesta:	El sistema se comporta estable ante un número elevado de peticiones concurrentes sin que se vea afectado su rendimiento.	
Decisiones arquitectónicas	**Razonamiento**	
Decisión: uso de Clústeres y Virtualización. Arquitectura en capas.	Un sistema es “escalable” cuando al aumentarse su capacidad de cómputo gana proporcionalmente en rendimiento (trabaja con un volumen superior de información y satisface un mayor número de solicitudes por unidad de tiempo). El estilo multicapas permite organizar los componentes del sistema de forma jerárquica en capas que pueden ubicarse en distintos nodos físicos de la infraestructura. Esto disminuye el riesgo que representa colocar en una única unidad de procesamiento todas las funcionalidades que engloba el sistema, evitando sobrecargas que lo hagan caer. El establecimiento de un clúster de aplicaciones (que contenga las capas de presentación y negocios) acompañado de otro para el manejo de los datos, (capas de acceso a datos y datos) -ambos equipados con balanceadores de carga- constituye una vía aconsejable para gestionar los recursos tecnológicos de una plataforma de altas prestaciones. Lo anteriormente explicado, unido a la función desempeñada por la tecnología de virtualización, permite hacer un uso más eficiente de la memoria, procesadores y dispositivos de almacenamiento contables, impactando positivamente sobre el rendimiento del sistema.	

Representación arquitectónica.		

3.1.2 Resultados obtenidos.

Del análisis exhaustivo de los 9 escenarios concebidos se obtuvieron los siguientes datos:

Relaciones entre atributos:

- ✓ El uso de la memoria caché del framework Symfony permite a los sistemas ganar en términos de velocidad de respuesta a las peticiones, a expensas del sacrificio de espacio de almacenamiento.
- ✓ El rendimiento máximo que pueden alcanzar los sistemas es delimitado por el nivel de escalabilidad que soporta la infraestructura tecnológica de la cual se disponga.

Puntos sensibles:

- ✓ Si en la plataforma utilizada el intérprete de JavaScript presente no implementa el estándar ECMAScript 262 la implementación de la capa de presentación podría no ser portable teniendo en cuenta las tecnologías consideradas para su composición. Esto puede mitigarse incluyendo en las especificaciones del sistema un listado de los navegadores que sean compatibles con el mismo.
- ✓ No todos los navegadores y sus versiones correspondientes implementan los estándares SVG y VML, responsables de la representación de gráficos vectoriales en documentos HTML. Esto puede mitigarse en el desarrollo de las aplicaciones mediante el uso o implementación de librerías que soporten ambos estándares.

- ✓ La interoperabilidad del sistema con aplicaciones externas que no hagan uso de la arquitectura propuesta puede verse afectada por problemas de incompatibilidad.

Riesgos:

- ✓ Una cantidad de código excesiva del lado del cliente puede afectar el sistema, demorando considerablemente la carga inicial del mismo. Esto puede mitigarse implementando técnicas de carga tardía y/o minimizando y compactando el código.

No Riesgos:

- ✓ La arquitectura propuesta es segura ante intentos de ataques por fuerza bruta, inyección de código SQL o scripts cruzados.
- ✓ La arquitectura propuesta posee mecanismos que permiten interactuar con otras aplicaciones web orientadas a servicios pertenecientes a sistemas externos.
- ✓ La arquitectura propuesta es confiable con respecto a la integridad y disponibilidad de los datos.
- ✓ La arquitectura propuesta considera técnicas que reducen el consumo de recursos y mejoran el rendimiento.
- ✓ La arquitectura propuesta es escalable, ofreciendo un desempeño adecuado ante volúmenes de transacciones elevados.
- ✓ La arquitectura propuesta es extensible, facilitando la introducción de nuevas funcionalidades al sistema con un impacto mínimo para su funcionamiento.
- ✓ La arquitectura propuesta es multiplataforma.

3.2 Conclusiones del capítulo.

El desarrollo del presente capítulo permitió llevar a cabo un análisis con el cual se demostró el ajuste de la arquitectura propuesta con respecto a los atributos de calidad requeridos. Para ello se hizo uso del método de evaluación de arquitecturas ATAM, el cual arrojó además los riesgos, no riesgos y puntos sensibles presentes en la arquitectura, lo cual favoreció la obtención de un mayor grado de refinamiento para la misma.

Conclusiones generales.

Al término de este trabajo se ha dado cumplimiento a los objetivos generales y específicos para los cuales fue concebido, mediante el desarrollo de las tareas en las cuales fueron desglosados cada uno.

- ✓ El estudio del modelo de negocio utilizado y las tecnologías asociadas a su implementación permitió definir un entorno tecnológico caracterizado por el uso de herramientas profesionales, pertenecientes a las vertientes del software libre, en correspondencia con las políticas establecidas por el país y la Universidad.

- ✓ Mediante las vistas arquitectónicas y artefactos conformados, se describe una Arquitectura de Software genérica que da las pautas a seguir para el desarrollo de una línea de productos de gestión empresarial de alta calidad y con gran aceptación en el mercado.

- ✓ Al aplicarse el método de evaluación ATAM a la arquitectura quedó confirmado cómo las decisiones tomadas, los estilos y patrones arquitectónicos utilizados y las tecnologías propuestas satisfacen en conjunto los Atributos de Calidad analizados.

Recomendaciones.

De forma general, la propuesta arquitectónica realizada responde a toda una gama de productos informáticos orientados a servicios a través de la web. A modo de establecer un proceso de mejoras continuo sobre la misma, se recomienda:

- ✓ Aplicar la Arquitectura de Software propuesta en el Departamento de Integración de Soluciones para el desarrollo de sistemas de gestión empresarial.
- ✓ Mantener esta propuesta en constante refinamiento, mediante la inclusión de nuevas tecnologías emergentes o versiones superiores de los programas aquí descritos que eleven la funcionalidad y calidad de los productos a desarrollar.
- ✓ Implementar un portal de servicios con una pasarela de pagos habilitada (necesaria para potenciar la gestión empresarial), dadas las características del modelo de negocio adoptado.

Referencias Bibliográficas.

1. **Osterwalder, Alexander y Pigneur, Yves.** *Business model generation: a handbook for visionaries, game changers and challengers.* New Jersey: s.n., 2009. 9782839905800 2839905809 9780470876411 0470876417.

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

3. **Perry, Dewayne y Wolf, Alexander.** *Foundations for the study of software architecture:* s.n., 2002.

4. **IEEE.** *Estándar 1471-2000.*

5. **Clements, Paul y Shaw, Mary.** *A field guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems:* s.n., 1996.

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

7. **Garlan, David y Shaw, Mary.** *An introduction to software architecture:* s.n., 1994. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21.

8. **Marks, Eric y Bell, Michael.** *Service-Oriented Architecture: a Planning and Implementation guide for Business and Technology:* s.n., 2006.

9. **Earl, Tomas.** *SOA, Principles of Service Design:* s.n., 2008.

10. **Robert, Armando.** *Arquitectura de Software para el Sistema Integrado de Gestión Estadística 2.0 Nuragas.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2009.

11. **Reynoso, Carlos y Kicillof, Nicolás.** *Lenguajes de Descripción de Arquitectura (ADL).* Buenos Aires: s.n., 2004.

12. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico 5ta Edición*: s.l.: McGraw-Hill, 2002. 8448132149.

13. **Camacho, E., Cardeso, F., Nuñez, G.** *Arquitecturas de Software. Guía de Estudio:* s.n., 2004.

14. **Losavio, Francisca y otros.** *ISO quality standards for measuring architectures.* Universidad de Venezuela: s.n., 2003.

Bibliografía.

1. **Alba, Julio.** (167, feb-mar 2008). *¿Qué es SOA?* Revista Bit.

2. *Apache Foundation, sitio oficial [en línea] http://www.apache.org.*

3. **Avila, Eladio.** *Propuesta de Arquitectura de Software para el desarrollo de portales web orientados al Ajedrez.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2009.

4. *Bacula, sitio oficial [en línea] http://www.bacula.org.*

5. **Barrios, Joel.** *Implementación de Servidores con GNU/Linux:* s.n., 2011.

6. **Betancourt, Adonys.** *Pautas para el diseño de la Vista de Infraestructura de una Arquitectura de Software de Gestión.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010.

7. **Bianco, P., Kotermanski, R., Merson, P.** *Evaluating a Service-Oriented Architecture.* Carnegie Mellon SEI: s.n., 2007.

8. **Buyya, R., Broberg, J., Goscinski, A.** *Cloud Computing. Principles and Paradigms:* s.l.: John Wiley & Sons, 2011. 978 0 470 88799 8.

9. *Cacti, sitio oficial [en línea] http://www.cacti.net.*

10. **Carrascoso, Y., Chaviano, E., Céspedes, A.** *Procedimiento para la Evaluación de Arquitecturas de Software Basadas en Componentes.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2008.

11. **Eclipse y otros contribuidores** *OpenUp v1.5.0.1 2004, 2008*.

12. **Eguíluz, Javier.** *Introducción a AJAX:* s.n., 2008.

13. **Feria, Liset.** *Vista de arquitectura de seguridad del proyecto ERP-Cuba.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010.

14. **Fowler, Martin y otros.** *Patterns of Enterprise Aplication Architecture:* s.l: Addison Wesley, 2002. 0-321-12742-0.

15. **Gamma, Erich y otros.** *Design Patterns. Elements of Reuseable Object-Oriented Software:* s.n., 1994.

16. **Hassan, Qusay F.** (en-feb 2011) *Demystifying Cloud Computing.* Revista Crosstalk. Mansoura University, Egypt.

17. **Hilley, David.** *Cloud Computing: A taxonomy of Platform and Infrastructure-level Offerings.* Georgia Institute of Technology: s.n., 2009.

18. **Hurwitz, Judith y otros.** *Cloud Computing for DUMMIES:* s.l.: John Wiley & Sons, 2010. 978 0 470 48470 8.

19. **Kabir, Mohammed J.** *La Biblia del Server Apache*.

20. **Kazman, R., Klein, M., Clements, P.** *ATAM: Method for Architecture Evaluation.* Carnegie Mellon SEI: s.n., 2000.

21. **Lazo, René y otros.** *Modelo de referencia para el desarrollo arquitectónico de sistemas de gestión. Proyecto ERP-Cuba.* Universidad de las Ciencias Informáticas, La Habana, Cuba: XVI Fórum de Ciencia y Técnica.

22. **Linthicum, David S.** *Cloud Computing and SOA Convergence in Your Enterprise. A Step-by-Step Guide:* s.l.: Addison Wesley, s.n., 2010. 0 13 600922 0.

23. *Manual de Symfony v2.0.1:* s.n., 2011.

24. **Martínez, Annierys y Pagán, Hector D.** *Propuesta de arquitectura para Juegos en Línea sobre plataforma web.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010.

25. **McGovern, James y otros.** *Enterprise Service Oriented Architectures. Concepts, Challenges, Recommendations:* s.l: Springer, s.n., 2006.

26. **Meriño, Eilen y Quiala, Marlies.** *Modelos de Negocio para comercializar productos del Grupo de gestión de Información Biomédica en la Universidad de las Ciencias Informáticas.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010.

27. *Microsoft Corporation, sitio oficial [en línea] http://www.microsoft.com.*

28. **Morales, L., Rodríguez, E., Rojas, S.** *Sistema de Requisiones y Órdenes de Compra. Documento de Arquitectura del Sistema.* Universidad de los Andes: s.n., 2008.

29. *MRTG, sitio oficial [en línea] http://www.mtrg.org.*

30. *Nagios, sitio oficial [en línea] http://www.nagios.org.*

31. *NetBeans IDE, sitio oficial [en línea] http://www.netbeans.org.*

32. *Portal de Producción UCI, Línea base de Arquitectura Proyectos UCI [en línea, citado el 10 de marzo del 2012] http://gespro.dgp.prod.uci.cu/projects/publico/wiki/Arquitectura_de_software.*

33. *PostgreSQL, sitio oficial [en línea] http://www.postgresql.org.*

34. *Ruby, sitio oficial [en línea] http://www.ruby-lang.org.*

35. **Santana, Marisel y Batista, Reinier.** *Propuesta de arquitectura Orientada a Servicios para alasMEDIGEN. Sistema Informático de Genética Médica.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010.

36. **Scalone, Fernanda.** *Estudio comparativo de los modelos y estándares de Calidad del Software.* Universidad Tecnológica Nacional, Facultad Regional Buenos Aires: s.n., 2006.

37. **Silega, Nemury.** *Guía metodológica para gestionar la integración de componentes en los proyectos del sistema CEDRUX.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010. TD-03933-11.

38. *Symfony Framework, sitio oficial [en línea] http://www.symfony-project.org.*

39. **TrowBridge, David y otros.** *Enterprise Solution Patterns Using Microsoft .Net:* s.n., 2003.

40. *Visual Paradigm, sitio oficial [en línea] http://www.visual-paradigm.com.*

41. *Webmin, sitio oficial [en línea] http://www.webmin.com.*

42. *Zend framework, sitio oficial [en línea] http://framework.zend.com.*

Glosario de términos.

1. Peer-to-peer: red de pares, conocida popularmente como red de punto a punto.

2. Browser: un browser o navegador web es un software cuya principal funcionalidad es interpretar el código en que están escritas las páginas web a fin de que el usuario pueda interactuar con las mismas.

3. Framework: herramienta informática que provee una infraestructura conceptual y tecnológica a partir de la cual pueden desarrollarse proyectos (de software) con bases similares.

4. IDE: acrónimo de *Integrated Development Enviroment* o *Entorno de Desarrollo Integrado,* al español. Un IDE es un programa informático compuesto por varias herramientas de programación (editor de texto, compilador, depurador, constructor de interfaces gráficas) que permiten a un desarrollador escribir aplicaciones en uno o más lenguajes de programación.

5. Mac: Mac o Mac OS, hace referencia al sistema operativo utilizado por la compañía Apple para su línea de ordenadores Macintosh.

6. ORM: el *Mapeo de Objetos Relacional* es una técnica de programación a través de la cual se pueden transformar los datos manipulados con un lenguaje de programación orientado a objetos para que puedan ser leídos en una base de datos relacional.

7. CEO: acrónimo del inglés empleado para referirse a oficiales de alto rango ejecutivo en un entorno empresarial, dígase jefe ejecutivo, presidente ejecutivo, director ejecutivo, entre otros.

8. CASE: acrónimo de *Computer Aided Software Engineering* o *Ingeniería de Software Asistida por Computadora.* El término responde a un conjunto de herramientas con cuyo uso es posible aumentar la productividad en un ambiente de desarrollo informático.

9. Ingeniería inversa: proceso mediante el cual se pueden conocer de un producto determinado, las tecnologías, métodos y herramientas utilizados para su composición.

10. Ofuscación: técnica que se aplica al código de un programa informático para dificultar la obtención de datos sensibles del mismo por aplicación de ingeniería inversa.

11. Bugs: vocablo del idioma inglés empleado en el campo de la informática para hacer referencia a fallos o defectos de software.