Universidad de las Ciencias Informáticas
Facultad 1

SIB: Arquitectura e integración de subsistemas.

Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autor: Fernando Nodarse García.

Tutores: Ing. Johann Rodríguez Hernández.

Ing. Juan Carlos Suárez López.

Ciudad de la Habana, junio del 2011.

"Año 53 de la Revolución Cubana"

"Si buscas resultados distintos,

no hagas siempre lo mismo."

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor del trabajo titulado: “Integración del Sistema de Integración Bancaria”, y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de ___ de 2011.

Fernando Nodarse García

Autor

Johann Rodríguez Hdez

Tutor

Juan Carlos Suárez López

Tutor

A mis padres, que tanto han hecho por mi.

A mi hermana, que es mi reflejo.

A mis segundos padres, mis abuelos.

A mi familia, por ser tan maravillosa conmigo.

Agradecimientos

A mis padres, gracias a ellos el camino se ha hecho más fácil.

A mi familia que siempre me ha ayudado.

A mi novia y su familia por como me han acogido y por los momentos compartidos.

A la Revolución Cubana y la Universidad de Ciencias Informáticas por brindarme la posibilidad de formarme como profesional.

A mis tutores, por dejarme esta oportunidad para mi solo.

A Johan por los caminos abiertos.

A Elizabeta, que por poco se vuelve a graduar con mi documento de tesis.

A Tony y Enca, pobre de mi tesis si no hubiese caído en sus manos, y por su aporte a mi español y los momentos del ajedrez.

A todos los profesores que me han impartido clases en especial a aquellos que más despertaron mi interés por el conocimiento: el Papujo, Arsenio, Yanelvis, María, mi profesora de matemática de la vocacional y mi profesor de física de la vocacional.

Al equipo del SIB, por el aporte a mi trabajo y por hacer amenos los días compartidos.

A Geidis por la ayuda brindada en estos años.

A Yamila y a Dairy por su aporte cuando me hizo falta.

A mis ensayadores personales Eduardo, Wilson y Osmany.

Al equipo de Pasaporte el cual tuvo que ver con una nueva etapa de mi formación, en especial a: Lázaro e Ireysi, por su ayuda durante toda la estancia

en la UCI y por recordarme las reuniones, jeje; a Erick que siempre ha aconsejado mis labores; y a Sotes por las respuesta e interrogantes a mis dudas.
A mis tribunales de tesis por sus constructoras recomendaciones.
A todos mis compañeros de aula, desde la primaria hasta quinto año de la UCI, por los momentos vividos.

RESUMEN

Los sistemas informáticos constituyen en la actualidad uno de los pilares fundamentales para el mundo empresarial, con ellos se llevan a cabo la gestión de los procesos y los recursos, la comunicación e intercambio de información entre las diferentes instituciones y muchas otras tareas. Es por ello que se requiere de sistemas robustos, seguros y confiables, lo cual se logra a partir del correcto diseño de la arquitectura de los mismos. La arquitectura de un software define la estructura de cada uno de los componentes del sistema, así como de sus interrelaciones; determina de igual forma su comportamiento y desarrollo a lo largo del tiempo.

En la República Bolivariana de Venezuela el Servicio Administrativo de Identificación, Migración y Extranjería (SAIME) realiza a diario números trámites para la identificación de los ciudadanos, así como el control migratorio y de extranjeros en el país. Para la realización de los mismos se hace necesario del pago a través de un comprobante emitido por alguna de las sucursales bancarias, lo que precisa de un alto nivel de confiabilidad de la información. En la actualidad no existe comunicación entre el sistema del SAIME y el de cada una de las sucursales de sus bancos recaudadores, por lo que no se puede validar la información emitida por los bancos.

Por esta situación se propone una arquitectura de software para el Sistema de Integración Bancaria (SIB), que permite la comunicación con el Sistema de Oficinas del SAIME y se integra con sistemas paralelos desarrollados por los bancos recaudadores. La propuesta de solución garantiza la seguridad e integridad de los datos a partir de un mecanismo de validación de la información emitida para cada uno de los pagos efectuados. Además, con la misma se logra que el SIB preste servicios a clientes que accederán desde Internet, con el fin de efectuar el pago de los trámites que sean de su interés.

Palabras claves: arquitectura de software, Sistema de Integración Bancaria, bancos recaudadores

Índice

Índice

INTRODUCCIÓN

En la actualidad los países se encuentran en una carrera contra el tiempo por transformar sus economías y adaptarlas a las condiciones que impone un mundo globalizado. Estas economías son cada vez más flexibles, permitiendo el comercio e inversiones dentro y fuera del marco de cada país; en donde el volumen de transacciones comerciales y financieras alcanza cantidades significativas. Todo este acelerado proceso se ha visto influenciado por un mayor desarrollo tecnológico y de las comunicaciones.

La globalización trae consigo mayores oportunidades de crecimiento económico para los países, pero también implica mayores riesgos por exposición a sus participantes, lo cual representa un reto para mejorar y desarrollar los sistemas de pago en cada una de las naciones.

Los sistemas de pago juegan un papel fundamental en una economía porque se consideran un elemento vital de la infraestructura financiera, posibilitando y facilitando la ejecución eficiente de las transacciones económicas, originando, por ende, ventajas competitivas que generan un ambiente propicio para la inversión. Son además un medio para promover la eficiencia económica al brindar mayor seguridad al proceso de intercambio de bienes, con menores costos de transacción, mayor rapidez y garantía de liquidez.

Por otra parte con el surgimiento de Internet se desarrollan nuevas formas de pagos, las cuales aunque novedosas no están exentas de problemas. Una de las tendencias actuales en el área de la arquitectura de software es crear sistemas con estos fines; que logren flexibilidad, mayor seguridad a la hora de efectuar las transacciones, sean adaptables, entre otras características.

En Venezuela este tema del pago electrónico se evidencia en cada una de las empresas y negocios particulares que tienen convenios o afiliaciones con sucursales bancarias a lo largo y ancho de todo el país. El Servicio Administrativo de Identificación, Migración y Extranjería (SAIME[1]) a diario realiza numerosos trámites, los que requieren de transacciones bancarias para el pago de los mismos. Este servicio de pago actualmente tiene elementos negativos y

[1]Organismo adscrito al Ministerio del Poder Popular para las Relaciones Exteriores y Justicia de Venezuela.

restricciones por lo que esta institución se da a la tarea de transformarlos para brindar mayor seguridad y fiabilidad en el pago de sus clientes.

Actualmente cuando un cliente del SAIME se dispone a realizar un trámite, se dirige a uno de los bancos recaudadores de dicha institución, deposita un monto y recibe un voucher[2] por cada trámite que desea le realicen. Este comprobante de pago lo presenta en las oficinas del SAIME como prueba de que tiene fondo para iniciar los trámites de su interés. Ante esta situación, no hay una manera efectiva de comprobar la legitimidad del voucher.

Partiendo de la necesidad del SAIME de viabilizar el proceso de notificación de pago bancario entre esta institución y sus bancos recaudadores, el Centro de Identificación y Seguridad Digital de la Universidad de las Ciencias Informáticas propone implementar un sistema que le garantice la fiabilidad y seguridad de los pagos efectuados por los clientes en las sucursales de sus bancos recaudadores.

El sistema prestará servicios a clientes que accederán desde Internet con el fin de efectuar el pago de los servicios que sean de su interés. Tendrá comunicación con los sistemas legados del SAIME y debe integrarse con sistemas paralelos desarrollados por los bancos recaudadores para obtener la información de los pagos efectuados en sus sucursales, garantizando la seguridad e integridad de los datos en todo momento.

Dada esta situación surge el siguiente **problema a resolver**: ¿Cómo desarrollar una arquitectura de software que garantice la robustez y flexibilidad del Sistema de Integración Bancaria para la Dirección de Gestión Administrativa del SAIME?

Teniendo en cuenta el problema planteado se define como **objeto de estudio**: La arquitectura de software. Por lo que se especifica el siguiente **campo de acción**: La arquitectura de software para el Sistema de Integración Bancaria de la Dirección de Gestión Administrativa del SAIME.

[2] Comprobante que emiten los bancos recaudadores cuando un cliente efectúa un depósito en las cuentas recaudadoras de SAIME.

Dadas estas condiciones se plantea como **objetivo general:** Desarrollar una arquitectura de software para el Sistema de Integración Bancaria de la Dirección de Gestión Administrativa del SAIME.

A partir del análisis realizado podemos plantear la siguiente **idea a defender**: Con el desarrollo de la arquitectura de software para el Sistema de Integración Bancaria, de la Dirección de Gestión Administrativa del SAIME, se garantizará la robustez y flexibilidad entre sus subsistemas.

Para dar cumplimiento al objetivo propuesto se proponen las siguientes **tareas de investigación:**

- Investigación sobre las tecnologías relacionadas con la integración de sistemas.
- Identificación de la metodología a utilizar.
- Caracterización de las herramientas a utilizar.
- Definición de las funcionalidades de los subsistemas y sus componentes asociados.
- Definición de las políticas y mecanismos para la comunicación con sistemas externos.
- Identificación de los protocolos de comunicación.
- Creación de niveles de accesibilidad.
- Caracterización de los elementos arquitectónicos dentro del sistema.
- Selección del equipamiento necesario para el ambiente de despliegue.
- Evaluación de la arquitectura propuesta.

A partir del análisis realizado podemos plantear la siguiente **idea a defender**:

- Con el desarrollo de una línea base para el Sistema de Integración Bancaria se garantizará la robustez y flexibilidad entre sus subsistemas.

Para el desarrollo de las tareas de investigación los **métodos científicos** utilizados son los siguientes:

- Histórico–Lógico: Permite realizar un estudio histórico de la evolución de la arquitectura y la integración de sistemas, con un orden lógico para identificar los principales hitos alcanzados en esta área y tendencias actuales para la solución de los problemas.
- Analítico–Sintético: Se utiliza para analizar la información y la documentación sobre las diversas tecnologías para el desarrollo del software, recopilada en las diversas fuentes identificadas.
- Modelación: Con el uso de este método se modelan los subsistemas en diagramas para darle seguimiento a través de las vistas arquitectónicas del sistema. De esta forma se pueden tener identificadas zonas críticas para reducir riesgos en la integración.

El presente trabajo está **estructurado** en 3 capítulos:

- Capítulo 1: Fundamentación Teórica, en el capítulo se abordan aspectos relacionados con las tendencias actuales de los sistemas de pago electrónico. Se especifican diferentes estilos arquitectónicos, así como mecanismos de integración que se utilizan para el desarrollo de sistemas de software. Además, se describen las herramientas, lenguajes y metodología empleados para realizar el presente trabajo de diploma.
- Capítulo 2: Arquitectura, en este capítulo se describen un conjunto de vistas que recogen los elementos arquitectónicamente significativos, además se desglosa la arquitectura en el diseño de los subsistemas, el ambiente de desarrollo, nombre de recursos y mecanismos de colaboración.
- Capítulo 3: Evaluación de la Arquitectura, se presenta la metodología de evaluación y diseño que se utiliza en el trabajo de diploma, así como los resultados obtenidos una vez que se aplicaron los parámetros para la validación del sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En la actualidad existe un auge en las Tecnologías de la Informática y las Comunicaciones (TIC) que trae aparejado la modernización de los sistemas informáticos, imprescindible para el desarrollo de la sociedad en su conjunto. Estos sistemas requieren a su vez de diseños arquitectónicos que garanticen alta disponibilidad, robustez y seguridad, entre otros aspectos. Para lograr las especificaciones de los clientes hay que tener presente a la hora del desarrollo de un software las mejores herramientas, lenguajes y metodologías a emplear.

1.1. Sistemas de pago electrónico

Un “sistema de pagos” es el conjunto de instrumentos, procesos y canales de transferencia de fondos entre los distintos individuos de una economía, necesario para el desarrollo de la actividad económica. (1) Por tanto, en sentido amplio, es la infraestructura a través de la cual se moviliza el dinero en una economía.

Con el surgimiento de Internet se han desarrollado los sistemas de pago. Aunque mantienen el mismo principio, un sistema de pago electrónico realiza la transferencia del dinero entre el comprador y el vendedor en una compra-venta electrónica. Es por ello que esta forma de pago es una pieza fundamental en el comercio electrónico.

Como ejemplos de sistemas de pago electrónico nos encontramos las **pasarelas de pago** o **TPV-virtual** para el pago con tarjeta, los sistemas de **monedero electrónico** y los sistemas que se conectan directamente con la **banca electrónica** del usuario.

En el pago con tarjeta, la **pasarela de pago** valida la tarjeta y organiza la transferencia del dinero de la cuenta del comprador a la cuenta del vendedor.

El **monedero electrónico**, sin embargo, almacena el dinero del comprador en un formato electrónico y lo transfiere al sistema durante el pago. El sistema de pago valida el dinero y organiza la transferencia a la cuenta del vendedor. También existe la posibilidad de que el sistema de pago transfiera el dinero electrónico al monedero electrónico del vendedor actuando en este caso como un intermediario entre ambos monederos electrónicos.

El pago a través de la **banca electrónica** enlaza un número de operación o venta realizada en el comercio o tienda virtual con la cuenta bancaria del cliente en el mismo sitio del banco, lo que reduce el riesgo de fraude al no transmitir información financiera personal por la red.

La seguridad en el comercio electrónico, y específicamente en las transacciones comerciales, es un aspecto de suma importancia para proteger los intereses tanto de compradores como de vendedores, la que se resume en autenticidad, confiabilidad e integridad de la información que se maneja. Las amenazas contra la seguridad hallan su hueco en la red, es por ello que la información que viaja utilizando mecanismos de codificación y encriptación. Con este fin se utilizan protocolos de seguridad en los sitios web, como: SSL y SET que permiten que la información que viaja por la red solo pueda ser interpretada por el sistema del cliente y el del servidor.

Otro punto crítico es la seguridad interna de la organización la cual es más difícil de combatir o controlar y sin embargo es la que más se pasa por alto. La mejor manera de combatirla es con una combinación de políticas de seguridad estrictas y un esquema de acceso a información privilegiada que solo les permita el acceso a aquellas personas que deban tenerlo.

1.2. Arquitectura de software

La asociación de los términos Arquitectura y Software es relativamente nueva. Hasta los años 80 los diseñadores de software se centraban en el código, donde el diseño involucra algoritmos y estructuras de datos, los componentes con los que trabajan son las primitivas de los lenguajes de programación y los mecanismos de composición incluyen variables, registros, cierta lógica de control, funciones y procedimientos. Esto trajo como consecuencia el surgimiento de problemas a la hora de desarrollar software a gran escala. Muchos ingenieros e investigadores centraron su atención en los problemas de diseño del software. Surge la idea de separar el diseño de la implementación y se aprecia la necesidad de utilizar notaciones y técnicas especiales.

A partir de los años 80 se popularizan paradigmas como la programación modular, la organización del software en paquetes, la programación orientada a objetos y la programación concurrente. La aplicación de estos paradigmas hace posible la puesta en práctica de principios

y reglas de diseño cuya realización se hacía anteriormente muy difícil, pero siguen moviéndose en el nivel del código. El diseño del software, como el de cualquier sistema complejo, requiere conocer y seguir disciplinadamente una serie de reglas bien definidas que indiquen cómo ensamblar y conectar los diferentes elementos del sistema y cómo gestionar el desarrollo y evolución del mismo. Es entonces en la década de los 90 donde se supuso el verdadero nacimiento de la Arquitectura de Software. Las investigaciones sobre metodologías y notaciones, que arrancaron en la década anterior, se materializan en un grupo de publicaciones sobre el tema.

Desde entonces se han tomado muchas definiciones de la Arquitectura de Software, la mayoría muy parecidas, diferenciándose entre sí por el énfasis que ponen sobre los aspectos que quieren resaltar. Según D. Garlan and D. Perry, *"La arquitectura de un sistema software es la estructura de sus componentes, sus relaciones y los principios que gobiernan su evolución a lo largo del tiempo".* (2)

1.2.1. Estilos arquitectónicos

Un estilo de arquitectura es la descripción de un conjunto de componentes y patrones que describen la transferencia de control y datos entre los mismos. Un estilo define una serie de restricciones respecto a los tipos de componentes que pueden utilizarse y a las formas en que dichos componentes pueden ser relacionados. Un estilo de arquitectura viene generalmente definido por una combinación de patrones. Los patrones pueden ser vistos como una micro-arquitectura que define un estilo puro o primario. La frontera entre patrón y estilo no está clara en la literatura consultada. De esta forma se define a los estilos arquitectónicos como:

"Una abstracción de nivel más alto que los patrones, que define una codificación particular de elementos de diseño y combinaciones formales de los mismos, limitando su número y el tipo de sus posibles combinaciones y mecanismos de interacción."(2)

Estilos de Llamada y Retorno.

Esta familia de estilos es de las más generalizadas en los sistemas de gran escala ya que hace énfasis en el poder de adaptarse a las modificaciones y el ser escalable. Parte de esta familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a

procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas. Entre estos estilos se encuentran:

Modelo Vista Controlador (MVC).

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual. La separación entre la vista y el controlador puede ser secundaria en aplicaciones de clientes ricos, es por eso que muchos frameworks de interfaz implementan ambos roles en un solo objeto. (3)

En aplicaciones de Web, por otra parte, la separación entre la vista (el browser) y el controlador (los componentes del lado del servidor que manejan los requerimientos de HTTP) está mucho más definida. Estas características le permiten soportar múltiples vistas, y una fácil adaptación al cambio. El patrón introduce nuevos niveles de indirección y profundiza la orientación a eventos del código, por lo tanto, aumenta ligeramente la complejidad de la solución.

Arquitecturas en Capas

Garlan y Shaw, definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. (4)

1.3. Integración de sistemas

Para apoyar sus diferentes procesos de negocios, la mayoría de las empresas poseen un conjunto muy variado de sistemas de información y control. Por lo general, estos sistemas han sido desarrollados en momentos diferentes y usando tecnologías de software disímiles, en muchos casos incompatibles.

Por otro lado, es común que estos sistemas operen en equipos de computación de diferentes marcas o tecnologías y corran bajo sistemas operativos diferentes. La red constituye la plataforma tecnológica que permite la interconexión física y lógica de los equipos de

computación usando redes de área local y su conexión a redes de amplio alcance, particularmente Internet. Para dar solución al problema de la integración de sistemas informáticos han surgido variadas propuestas.

1.3.1. Integración de datos

La integración de datos entre dos aplicaciones involucra la transformación de los datos de una de ellas al formato usado por la otra, sin que sea necesario modificar el código de las aplicaciones. Estandarizar ficheros con el uso de XML (*Extensible Markup Language*) o CSV (*Comma-Separated Values*) son herramientas que se pueden usar en este sentido por su facilidad de representar datos e independencia de tecnologías informáticas.

Las bases de datos ofrecen también una vía para ello debido a la existencia de un lenguaje estándar universal para el acceso a base de datos (SQL, *Structured Query Language*). Las estrategias más utilizadas basadas en este enfoque son:

- Creación de una base de datos con información de otras bases de datos. Se basa en desarrollo de modelos de intercambio de información y representación común de la información.
- Utilización de procesos que alimentan la base de datos de una aplicación con información de otras bases de datos. Esto permite que una aplicación acceda a información de otras aplicaciones sin necesidad de ser modificada.
- Utilización de una base de datos común diseñada específicamente para ser accedida por múltiples aplicaciones.

1.3.2. Sistemas distribuidos

"Sistemas cuyos componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor".

Los sistemas de objetos distribuidos permiten la interacción entre objetos mediante un gestor de solicitudes de objetos, el cual ayuda a la aplicación cliente a localizar el objeto requerido. Sin embargo, el gestor no define un entorno que controle el objeto distribuido.

Algunas de las tecnologías de objetos distribuidos son:

- **RMI (*Remote Invocation Method)*:** Fue el primer *framework* para crear sistemas distribuidos de Java. El sistema de Invocación Remota de Métodos (RMI) de Java, permite a un objeto que se está ejecutando en una Máquina Virtual Java (VM) llamar a métodos de otro objeto que está en otra VM diferente. Esta tecnología está asociada al lenguaje de programación Java, es decir, que permite la comunicación entre objetos creados en este lenguaje.
- **DCOM (*Distributed Component Object Model)*:** El Modelo de Objeto Componente Distribuido está incluido en los sistemas operativos de Microsoft. Es un juego de conceptos e interfaces de programa, en el cual los objetos de programa del cliente pueden solicitar servicios de objetos de programas a servidores en otros ordenadores dentro de una red. Esta tecnología está asociada a la plataforma de productos *Microsoft*.
- **CORBA (*Common Object Request Broker Architecture*.)**. Tecnología introducida por el Grupo de Administración de Objetos OMG, creada para establecer una plataforma para la gestión de objetos remotos, independiente del lenguaje de programación.

1.3.3. Servicios Web

Los servicios Web son procedimientos remotos que son accedidos a través de la Web para el intercambio de datos entre aplicaciones, utilizando protocolos basados en XML. Generalmente se utiliza el protocolo SOAP3(*Simple Object Access Protocol*) en combinación con HTTP (*Hypertext Transfer Protocol*). Además de SOAP, se utiliza un lenguaje llamado WSDL4 (*Web Services Description Language*) para definir las interfaces programáticas de los procedimientos

[3] Protocolo de intercambio entre aplicaciones ejecutadas sobre cualquier plataforma. El llamado al servicio SOAP pone en marcha un flujo ASCII encerrado entre etiquetas XML y transportado en el protocolo HTTP.
[4] Da en formato XML la descripción de los servicios web precisando los métodos que pueden ser invocados, su firma y el punto de acceso (URL, puerto, etc…)

y UDDI (*Universal Discovery Description and Integration*), el cual permite localizar y acceder a los servicios.

La interoperabilidad se consigue mediante la adopción de estándares abiertos que permiten que ofrecen funcionalidades de forma independiente a la tecnología en la que se han desarrollado. Son realmente una forma de intercambiar información, de pedir algo sin tener en cuenta el lenguaje de programación; para dar un servicio de forma independiente solo hace falta saber cómo invocar al servicio.

1.4. Ambiente de desarrollo

1.4.1. Metodología de desarrollo de software : RUP

El Proceso Unificado es un proceso de desarrollo de software y como tal define un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. RUP es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, áreas de aplicación, tipos de organizaciones, niveles de aptitud y tamaños de proyecto.

El Proceso Unificado está basado en componentes y utiliza el Lenguaje Unificado de Modelado (UML). RUP en dos dimensiones representa el proceso en el que se grafican los flujos de trabajo y las fases, muestra la dinámica expresada en iteraciones y puntos de control. Divide en cuatro fases el desarrollo del software Inicio, Elaboración, Construcción y Transición, en cada una se obtiene un producto final y cada una está desarrollada mediante el ciclo de iteraciones, las cuales tienen como función la evaluación de las iteraciones precedentes. (5)

Características principales de RUP.

- Guiado por casos de uso.
- Centrado en la arquitectura.
- Iterativo e incremental.

Ventajas de RUP.

- Se basa en las mejores prácticas que se han intentado y se han probado en el campo de la ingeniería.
- Mitigación temprana de posibles riesgos altos.
- Progreso visible en las primeras etapas.
- Temprana retroalimentación que se ajusta a las necesidades reales.
- Gestión de la complejidad.
- Conocimiento adquirido en una iteración puede aplicarse de iteración a iteración.

1.4.2. Lenguajes utilizados

Java

Java ofrece toda la funcionalidad de un lenguaje potente, con características propias del paradigma de la orientación a objetos, pero sin las características menos usadas y más confusas de estos. Se diseñó para ser parecido a C y C++ y así facilitar un rápido y fácil aprendizaje. Realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución reduciendo en un 50% los errores más comunes de la programación con otros lenguajes. Es *multithreaded* permite muchas actividades simultáneas en un programa mejorando el rendimiento interactivo y el comportamiento en tiempo real. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. El código fuente Java se "compila" a un código de bytes de alto nivel independiente de la máquina. El hecho de que las aplicaciones de no se les permite el acceso a zonas delicadas de memoria o de sistemas, lo que hace que se evite la interacción de ciertos virus, y la seguridad le sea integrada en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario, hace que las aplicaciones de Java resulten extremadamente seguras. (6)

PHP

Creado originalmente por Rasmus Lerdorf en 1994, PHP (*Hypertext Pre-Processor*) se trata de un lenguaje de programación interpretado en el servidor, campo más tradicional y principal foco

de trabajo. Este lenguaje pretende brindar a los creadores de sitios webs la posibilidad de desarrollar sitios dinámicos en forma sencilla y rápida aplicando técnicas de programación orientada a objetos. Es interpretado en el lado del servidor, utilizado para la generación de páginas web dinámicas, similar al *Active Server Pages* (ASP) de *Microsoft*, embebido en páginas XHTML y ejecutado en el servidor. Puede ser utilizado en cualquiera de los principales sistemas operativos del mercado y soporta la mayoría de servidores web de hoy en día. Con PHP no se encuentra limitado a resultados en HTML, permite: creación de imágenes, archivos PDF e incluso películas Flash. También puede presentar otros resultados, como XHTML y cualquier otro tipo de ficheros XML. PHP puede autogenerar estos archivos y almacenarlos en el sistema de archivos en vez de presentarlos en la pantalla, creando un caché en el lado-servidor para contenido dinámico. Tiene capacidad de conexión con la mayoría de los Sistemas Gestores de Base de Datos que se utilizan en la actualidad, destacando su conectividad con MySQL y PostgreSQL. Con el crecimiento de PHP surgieron proyectos asociados, tales como *Frameworks*, *IDE's (*Entornos de desarrollo integrado) que le han dado al lenguaje una robustez y consistencia aún mayor. (7)

XHTML

HTML (*Extensible Hypertext Markup Language o* Lenguaje extensible de marcación de hipertexto). Este lenguaje ofrece la norma clásica para crear páginas web, el Lenguaje de marcación de Hipertexto (HTML, *Hypertext Markup Language*), y la nueva norma para datos descriptivos, XML.

Surge a partir de la necesidad de un sistema estándar para añadir nuevas características a las páginas. Desde fuera, XHTML es muy similar a XML, sus etiquetas, atributos y valores; el tener todas sus características le permiten que lo puedan entender todos los navegadores. Está diseñado para guiar el desarrollo de documentos bien formados y válidos. (8)

UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual, de propósito general para el modelado orientado a objetos, usado para especificar, visualizar, construir y documentar artefactos de un sistema de software. Es una consolidación de muchas de las notaciones y conceptos más usados en el mundo orientado a objeto.

Incluye aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. Divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto.

UML introduce nuevos diagramas que permiten tener una visión dinámica del sistema, lo que facilita detectar problemas de la estructura en la fase de diseño. Su utilización es independiente del lenguaje de programación y de las características de los proyectos. (9)

XML

XML es un subconjunto simplificado de SGML (*Standard Generalized Markup Language*). Es un metalenguaje, es decir: lenguaje para definir lenguajes de estructuración de texto basados en marcas explícitas o etiquetas, que son entendibles por una persona y que pueden ser interpretadas por un computador. De aquí parte su uso en los protocolos estándares para el intercambio de datos o especificación de metadatos. XML impone una sintaxis más rígida para las marcas, que permite su proceso de forma más eficiente.

Distingue entre minúsculas y mayúsculas. Las marcas se obtienen a medida que se interpreta el documento y permite definir lenguajes de marcas para cualquier fin y tiene capacidades de validación de datos. (10)

YAML

YAML es un formato para serializar datos, que es fácil de procesar por las máquinas, fácil de leer para las personas y fácil de interactuar con los lenguajes de script. Dicho de otra forma, YAML es un lenguaje muy sencillo que permite describir los datos como en XML, pero con una sintaxis mucho más sencilla. YAML es un formato especialmente útil para describir datos que pueden ser transformados en arreglos (*array*) simples y asociativos. YAML utiliza la tabulación para indicar su estructura, los elementos que forman una secuencia utilizan un guión medio y los pares clave/valor de los arreglo asociativos se separan con dos puntos. YAML también dispone de una notación resumida para describir la misma estructura con menos líneas. (11)

CSS

CSS (*Cascade Style Sheet*) especifica la forma del diseño de los documentos (tanto XHTML como HTML). Una misma página Web (un mismo documento XHTML, por ejemplo) puede ser vista de diferente forma en un PC, gracias a diferentes hojas de estilo CSS.

Con CSS se logra aumentar la densidad de las palabras clave dentro de los contenidos, ya que muchas de las etiquetas ocuparán muchísimo menos espacio. Esto también supone un menor peso para las páginas Web. CSS se puede lograr encerrar cada información dentro de la etiqueta que corresponda al significado semántico adecuado. El lenguaje de las Hojas de estilo, aunque es muy potente, es relativamente sencillo y fácil de aprender. (12)

1.4.3. Frameworks

Un framework es una estructura de software compuesta de componentes personalizables e intercambiables, a la que podemos añadirle piezas para construir una aplicación. Simplifica el desarrollo de un software mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, proporciona estructura al código fuente, forzando al desarrollador a crear códigos más legibles y más fáciles de mantener. Encapsula operaciones complejas en instrucciones sencillas.

Spring

Spring ofrece una amplia gama de capacidades para la creación e integración de aplicaciones empresariales desarrolladas en Java. Posee una potente gestión de configuración basada en JavaBeans, a través de un contenedor de Inversión de Control (IoC) que gestiona el ciclo de vida y la configuración de los objetos de la aplicación. El contenedor maneja objetos por el desarrollador y resuelve dependencias entre los objetos que contiene, por su naturaleza fomenta las buenas prácticas de programar orientado hacia interfaces más que a clases; lo que permite utilizar interfaces que servirán de abstracción para aquellas clases que hagan uso de los DAOS (Objeto de Acceso a Datos) y objetos de negocio. Provee una consistente capa genérica de abstracción para la gestión de transacciones a través de JTA (*Java Transaction API*), JDBC (*Java Database Connectivity*), JPA (*Java Persistence API*), JDO (*Java Data Objects*) e *Hibernate* sin necesidad de crear y cerrar sesiones. Permite la administración declarativa y programática de transacciones para que se realicen los cambios en la base de

datos o en caso de error que la base de datos quede en un estado consistente. Posee una capa de abstracción JDBC independiente del gestor de base de datos simplificando el manejo de errores, y reduce considerablemente la cantidad de código necesario. Se integra con varios ORM en términos de soporte a implementaciones DAOs y estrategias con transacciones, prestando especial soporte a *Hibernate* (13). Además es un proyecto de Software libre con abundante documentación.

Symfony

Symfony es un *framework* diseñado para optimizar el desarrollo de las aplicaciones web, separando la lógica de negocio, la lógica de servidor y la presentación de dichas aplicaciones. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo. Está desarrollado completamente con PHP 5 y es compatible con la mayoría de los gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft.

Características:

- Fácil de instalar y configurar en la mayoría de las plataformas (Compatible con Windows, Unix, Linux).
- Independiente del sistema gestor de bases de datos.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayoría de las mejores prácticas y patrones del diseño para la web.
- Preparado para aplicaciones empresariales, adaptable a las políticas y arquitecturas propias de cada empresa,
- Estable para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de php Documentor y que permite un mantenimiento muy sencillo.

- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros. (11)

Quartz

Quartz es un programador de tareas en Java. Para empezar, hay que decir que J2SE ya viene de serie con soporte para la programación de tareas. Aporta un framework mucho más completo, y con características interesantes que otros soportes de J2SE para la programación de tareas, como:

- Configuración declarativa (ficheros XML o properties), además de programática.
- Multitud de opciones de programación (por ejemplo, sintaxis tipo Cron).
- Mantenimiento de estado de las tareas.
- Potente gestión de errores o tareas incompletas recuperables.
- Disponibilidad de *Listeners* no intrusivos sobre las tareas, los disparadores o sobre el propio programador.
- Integración con aplicaciones web vía *Servlet* o *ContextListener*.
- Integración con Spring.
- Soporte a entorno distribuido (clúster).
- Lanza un hilo por cada tarea que se ejecuta.

Es muy común la necesidad de dar de alta procesos periódicos asociados a una aplicación web. Quartz es especialmente útil, precisamente porque elimina la obligación de tener crones de Linux o tareas programadas de Windows, de forma que toda nuestra aplicación, con la funcionalidad completa, reside en el WAR. (14)

1.4.4. Servicios web

Axis2

Axis, realizado por *Apache Software Foundation*, permite la creación y el despliegue de servicios Web. Es un paquete Java libre que provee:

- Un entorno que puede funcionar como un servidor SOAP.
- Una API para desarrollar Servicios Web.
- El soporte para diferentes capas de transporte: HTTP, FTP, SMTP, POP et IMAP.
- La serialización/deserialización automática de objetos Java en mensajes SOAP.
- Herramientas para crear automáticamente WSDL correspondientes a clases Java o inversamente para crear clases Java sobre la base de un WSDL.
- Herramientas para desplegar, testear y monitorear servicios Web.

Axis 2.0 es una nueva versión que tiene como propósito ser más eficaz, modular y orientada a XML que la versión precedente. Utiliza su propio modelo de objeto que analiza para mejorar la velocidad de procesamiento. Permite despliegue en caliente, es decir los nuevos servicios se pueden agregar al sistema sin tener que cerrar el servidor. Provee un cierto número de módulos relacionados con la seguridad, las transacciones, etc. (15)

1.4.5. Mapeador de Objetos Relacional(ORM)

Cuando se trabaja con programación orientada a objetos y bases de datos relacionales se está en presencia de paradigmas diferentes. El modelo relacional trata con entidades, relaciones, tuplas y conjuntos. Por su parte el paradigma orientado a objetos trata con objetos, sus atributos y relaciones entre los mismos. Cuando se necesita que los objetos sean persistentes se hace uso de las bases de datos y para relacionarla con las clases creadas para esos objetos, se utilizan herramientas que mapean estas entidades persistentes y crea las asociaciones. Un ORM contribuye a evitar esta diferencia entre ambos conceptos.

Doctrine

Doctrine es un ORM para PHP que se sitúa arriba de la poderosa PHP DBAL (Capa de Abstracción de la Base de Datos). Es una librería muy completa y muy configurable, con ventajas en el rendimiento de ejecución sobre otros ORM que se integran con Symfony. Permite la generación automática del modelo de la base de datos que establece la relación entre un modelo relacional y un modelo de clases. Esto se puede hacer con código PHP o con

YAML, que es un formato de serialización de datos fácil de interpretar y muy usado para este fin. Ofrece clases que nos facilita métodos para insertar, actualizar o eliminar registros, entre otros. Todas las consultas son en un lenguaje OO (Orientado a Objetos) llamado DQL (*Doctrine Query Language*). Esta brinda a los desarrolladores una poderosa alternativa al SQL, manteniendo una máxima flexibilidad sin necesidad de duplicar código.

Hibernate

Hibernate es una herramienta para la plataforma Java que parte de una filosofía de mapear objetos Java ¨normales¨, también conocidos como "POJOs" (*Plain Old Java Objects*), para establecer una relación directa entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML), diseñado para ser editado a mano, o con el uso de anotaciones que permiten establecer estas relaciones. Estos objetos y ficheros son generados por el propio Hibernate y aunque necesitan modificaciones que optimicen su funcionamiento son estables y fáciles de mantener. Los objetos persistentes podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. Se pueden establecer asociaciones de uno a muchos, de uno a uno, de muchos a muchos y hasta asociaciones de herencia. Permite implementar la capa de persistencia de forma sencilla, optimizando el acceso a los datos. Es independiente del SGBD y del dialecto SQL, proporcionando su propio lenguaje de consultas. Por lo que no es necesario gestionar la conexión de apertura y cierre con la base de datos y las consultas no tiene porqué realizarse con SQL para ello se utilizan HQL (*Hibernate Query Language*) o su propias API, para construir consultas programáticas, conocidas como criteria, las cuales son más sencillas e intuitivas de emplear y reducen fallas al crear código SQL.

1.4.6. Sistema Gestor de Base de Datos

Se trata de un conjunto de programas que tratan de establecer y mantener las trayectorias de acceso a la base de datos, de tal forma que los datos puedan ser localizados y accedidos rápidamente. Maneja los datos de acuerdo a las peticiones de los usuarios.

PostgreSQL

PostgreSQL es un sistema gestor objeto-relacional, ya que incluye características de la orientación a objetos, como pueden ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. (17)

Ventajas:

- Aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas.
- Usa una arquitectura proceso por usuario cliente/servidor: Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectarse a PostgreSQL.
- Altamente extensible: Soporta los tipos de datos base y otros como los de fechas, monetarios, elementos gráficos, datos sobre redes (MAC, IP), cadenas de bits, etc. Además operadores, funciones, métodos de acceso y tipos de datos definidos por el usuario.
- Integridad Referencial: Es utilizada para garantizar la validez de los datos de la BD.
- Tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. Además tiene habilidad para usar Perl, Python, o TCL como lenguaje procedural embebido.
- Control de Concurrencia Multi-Versión: Es usado para evitar bloqueos innecesarios, permite la lectura sin que sea bloqueada por los que escriben y actualizan registros.
- Incrementa la dependencia de la BD al registro de cambios antes de que estos sean escritos en la base de datos. Esto garantiza que en caso de que la BD se caiga, existirá un registro de las transacciones a partir del cual se podrá restaurar la BD desde el punto en que se quedó.
- Se ejecuta en la mayoría de los sistemas operativos, incluidos Linux, Unix y Windows.
- Tiene más de 15 años de activo desarrollo y arquitectura probada que se ha ganado una muy buena reputación por su confiabilidad e integridad de datos.

Desventajas:

- Consume muchos recursos y carga con facilidad el sistema.
- Velocidad de respuesta lenta al gestionar BD relativamente pequeñas, aunque esta misma velocidad la mantiene al gestionar BD realmente grandes. (17)

1.4.7. Herramienta de modelado

Visual Paradigm

Visual Paradigm es una herramienta de modelado que utiliza el lenguaje de modelado estándar UML, permite la generación de códigos e ingeniería inversa. Con una clase de diseño bien especificada, Visual Paradigm puede generar código hasta en 15 lenguajes de programación. Visual Paradigm actualmente cumple con las políticas de migración a Software Libre en Cuba, ya que es una herramienta multiplataforma que se puede utilizar tanto en Linux como en Windows. Tiene una interfaz muy intuitiva y es de fácil aprendizaje para los desarrolladores.

Permite la generación automática de diagramas a partir de descripciones de casos de usos. Permite además hacer descripción de los casos de usos dando una gran variedad de plantillas predeterminadas permitiendo personalizarlas. Combina las funcionalidades de todas las ediciones en una amplia plataforma de modelado visual. Visual Paradigm para UML facilita a los desarrolladores de software una herramienta vanguardia para crear aplicaciones de elevada calidad, rápidas y más baratas. Brinda una mejor interfaz gráfica al usuario. (18)

1.4.8. Herramientas para el desarrollo del sistema

Netbeans

El IDE NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Las funciones de este IDE son provistas por módulos donde cada uno provee una función bien definida. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso. El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Extiende las características existentes del Java

EE (incluyendo Soporte a Persistencia, EJB 3 y JAX-WS). Soporta el desarrollo de Aplicaciones empresariales con Java EE 5 y Java EE 6, incluyendo herramientas de desarrollo visuales de SOA, herramientas de esquemas XML y orientación a web servicies. Permite crear aplicaciones Web con PHP 5, un potente *debugger* integrado y además viene con soporte para Symfony.

Servidor Apache

El servidor Apache, es un servidor web muy usado por su excelencia, calidad de servicios, robustez y estabilidad, que hacen que día a día usuarios y servidores reiteren su confianza y renueven la elección a este servicio.

Características que posee Apache y que la llevó al éxito en la inserción y utilización en ámbitos empresariales, tecnológicos y educativos:

- Fundamentalmente corre sobre una multitud de plataformas y Sistemas Operativos.
- Apache ofrece tecnología libre y de código abierto, otorgándole una transparencia y dando la posibilidad de conocer qué es lo que realmente estamos instalando.
- Apache es un servidor Web altamente configurable y de diseño modular, capaz de ampliar su funcionalidad y calidad de servicios.
- Apache trabaja en conjunto con gran cantidad de Lenguajes de Programación interpretados como PHP (*Hypertext Pre-Processor*), Perl, soporte con CGI (*Common Gateway Interface*), Java, JSP (*Java Server Pages*) y otros lenguajes de script, el complemento ideal para los sitios web dinámicos que vemos en la actualidad.
- Es posible configurar y personalizar cada uno de los mensajes de error que se pueden producir por la utilización del servidor.
- Contar con los archivos *Logs*, en donde registra gran cantidad de información global del sistema, errores producidos en un determinado tiempo, en la cual estos archivos son de gran importancia para los administradores de sistemas y pueden influenciar de alguna manera las políticas de seguridad debido a la gran cantidad de información que contiene.

- Otra particularidad propia de Apache y que está muy ligada a su pensamiento y filosofía libre, es que al ser tan popular y utilizado, es posible encontrar gran cantidad de documentos, ejemplos y ayuda en Internet en todos los idiomas. (19)

1.5. Propuesta de solución

El SIB en su primera versión garantiza la fiabilidad y seguridad de los pagos efectuados por los clientes en las sucursales de sus bancos recaudadores. Para ello el sistema constará con una aplicación web a la que accederán los clientes para solicitar los servicios de su preferencia y con fines administrativos por parte del SAIME, dos aplicaciones de escritorio para comunicarse con los bancos, con el fin de conocer los pagos efectuados en sus sucursales, una base de datos centralizada para almacenar la información de los servicios, los clientes y sus pagos, y se publicarán servicios web con funcionalidades de interés para el Sistema de Oficinas del SAIME.

1.6. Conclusiones

El SIB forma parte de los sistemas de pago electrónico, pero con la particularidad de que ofrece a los clientes diferentes variantes para ejecutar los pagos que se realizan tanto en los bancos, como en los puntos de venta y cajeros electrónicos. Para ofrecer esta posibilidad y como medida de seguridad, la información es encriptada y de esta manera se garantiza la integridad de los datos que viajan a través de la red o de los voucher que portan los ciudadanos.

El Sistema de Oficinas del SAIME y el SIB deben integrarse mediante servicios web, pues no se han empleado las mismas tecnologías en los desarrollos, y precisamente los servicios web permiten el intercambio de información entre aplicaciones que utilicen diferentes lenguajes de programación. Para la integración del SIB con los bancos recaudadores del SAIME, se utilizarán ficheros XML que permiten conocer la información de los pagos.

Para el acceso de los clientes a través de Internet, el SIB contiene un subsistema web que permite conocer el estado de los trámites de pago y acceder a los servicios en línea que ofrece el SAIME.

CAPÍTULO 2: ARQUITECTURA

Introducción

La arquitectura de software es primordial para lograr sistemas informáticos robustos y seguros, constituye un modelo perceptible de la estructura del sistema y de las relaciones entre sus componentes. El nivel de abstracción necesario para describirla se sitúa aún muy lejos de los detalles de implementación, pero exhibe ya toda una serie de características que van a condicionar tanto el producto final como su proceso de desarrollo. Por ello, si la arquitectura está correctamente descrita, permite o favorece la documentación de la estructura y propiedades del sistema en un lenguaje perceptible por todas las partes implicadas en el desarrollo del mismo. De igual forma permite la evaluación de las decisiones de diseño desde las etapas más tempranas posibles del proceso de desarrollo y durante la evolución del mismo. Contribuye a la reutilización de modelos y componentes de software y su uso comercial, así como al prototipado evolutivo y el crecimiento incremental. (20)El SAIME realiza a diario números trámites para la identificación de los ciudadanos así como el control migratorio y de extranjeros en el país. Para el pago de los mismos y la comprobación de estos se necesita de un sistema informático, de ahí la necesidad de proponer una arquitectura que garantice robustez y flexibilidad al mismo.

2.1. Descripción del sistema

El sistema propuesto permite la comunicación entre el Sistema de Oficinas del SAIME y sus sucursales bancarias. Se encuentra dividido en 4 aplicaciones como se muestra en la Figura 1. Se utiliza una base de datos común que se encarga de integrar dichas aplicaciones a nivel de datos.

Solicitud de pago de servicio: Es una aplicación web que está de cara a Internet, facilitándole al cliente del proveedor del servicio acceder con la finalidad de efectuar los pagos de los servicios que sean de su interés.

Pagos: Ofrece un conjunto de servicios web (para mayor información sobre los mismos consultar epígrafe 2.3.1.) que serán consumidos por el Sistema de Oficinas del SAIME, para intercambiar información referente a los pagos de los ciudadanos.

Gestión de comunicación bancaria: Se encarga de la comunicación con los bancos, configurar las direcciones de los servidores SFTP de los respectivos bancos, para generar y notificar los archivos de seguridad.

Servicio de actualización de pagos: Permitirá gestionarlas lecturas que se ejecutarán sobre los servidores SFTP destinados para los bancos. Esta aplicación permitirá ejecutar lecturas sobre estos servidores y monitorear la ejecución automática de éstas.

La infraestructura de comunicación que se propone, entre las distintas entidades, es a través de una VPN. Los bancos deben proveer un servidor SFTP para alojar los diferentes archivos. Este servicio está propuesto que sea brindado por el propio banco para evitar problemas de actualización y, por ende, un mayor trabajo en su desarrollo de software, pues con ello se garantiza que de ocurrir alguna falla en las comunicaciones, el mensaje quede disponible para el momento en que se restablezcan las mismas y SAIME pueda actualizarse desde la publicación.

Figura 1 Vista del sistema.

Cada aplicación está dividida en módulos, para lograr reusabilidad y facilitar el mantenimiento del mismo ante cambios que se puedan realizar en los procesos de negocio.

Las aplicaciones **Servicio de actualización de pago** y **Gestión de comunicación bancaria** hacen uso de un módulo común en el cual se agrupan submódulos con funcionalidades comunes para ambos subsistemas (Autenticación, Notificación y CSS), como se muestra en la Figura 2. Estas aplicaciones a su vez también cuentan con módulos propios, GenerarCódigoSeguridad para el subsistema **Gestión de comunicación bancaria** y Lectura en el caso de **Servicio de actualización de pago**.

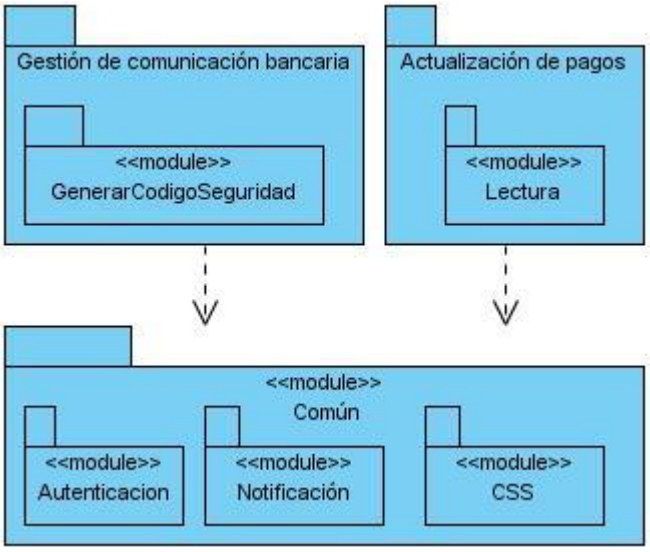


Figura 2 Módulo de las aplicaciones de escritorio.

Descripción de los módulos:

- **Autenticación:** Se validan los usuarios que accedan a la aplicación.
- **Notificación:** Gestión de las notificaciones y el envío de correos.
- **CSS:** Valida las ráfagas de los pagos.
- **Generar Código Seguridad**: Configurar la conexión con los servidores SFTP de los bancos y generar, enviar y notificar los ficheros de seguridad de cada banco.
- **Lectura**: Gestionar y ejecutar lecturas.

La aplicación **Solicitud de pago de servicio** se divide en dos aplicaciones y estas a su vez están formadas por módulos, como se muestra en la Figura 3. Las aplicaciones son:

frontend: Contiene los servicios que puede gestionar los clientes desde la interfaz pública. Consta de los siguientes módulos:

- **Cliente:** Gestionar las preferencias de los clientes.
- **Autenticación:** Se validan los usuarios que accedan a la aplicación.
- **Reportes:** Emisión de los reportes de la aplicación.
- **Solicitudes:** Gestión de las solicitudes de preferencia para los usuarios.
- **Devoluciones:** Gestionan las devoluciones emitida por los clientes.

backend: Contiene los servicios para la gestión de la aplicación en general. Consta solamente de un módulo:

- **Administración**: En el mismo se gestiona la información contenida en la aplicación (Nomencladores, Servicios que presta), además de los datos de los clientes y los servicios.

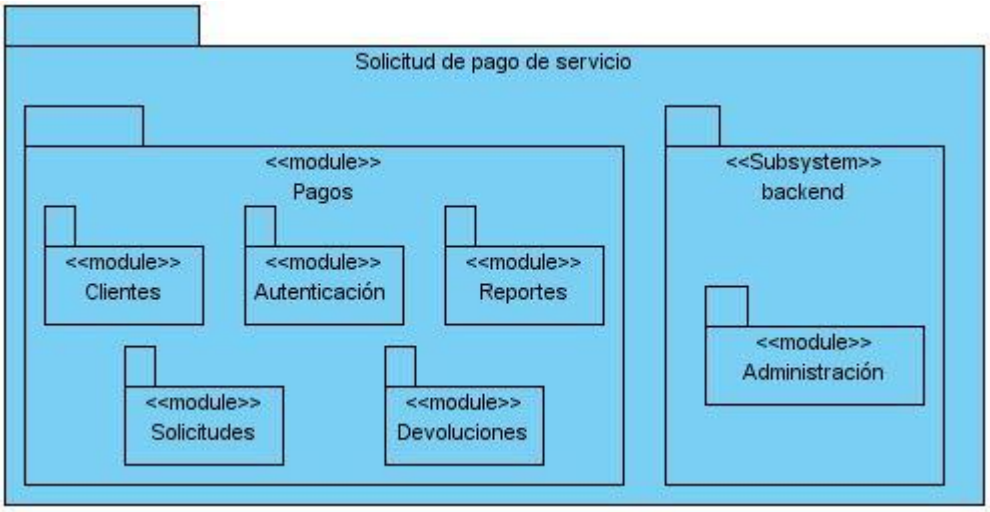


Figura 3 Módulos de la aplicación Solicitud de pago de servicios.

Aplicación **Pagos** cuenta solamente de un módulo como se muestra en la Figura 4**.**

Descripción del módulo:

- **Pagos:** Registrar los pagos de los clientes.

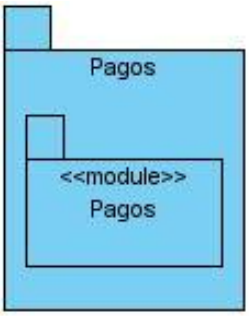


Figura 4 Módulo de la aplicación Pagos.

2.2. Requisitos no funcionales

La arquitectura de software (AS) está centrada en los requisitos no funcionales (RNF) de la solución. Estos RNF, se han de tener en cuenta para la implementación del sistema.

2.2.1. Requisitos de software.

- Sistemas operativos multiplataforma.
- Navegador web: Internet Explorer, Mozilla.
- Servidor gestor de base de datos PostgreSQL 8.4.

2.2.2. Restricciones en el diseño y la implementación.

- La herramienta de desarrollo de la aplicación será el *framework* Symfony, Netbeans, Visual Paradigm, Spring y como mapeo de objeto relacional Doctrine e Hibernate.
- El lenguaje de programación será PHP 5.3 y Java.
- Se utilizará la Programación Orientada a Objetos.
- Se intercambiará información con servidores SFTP.
- Se intercambiarán archivos en formato XML y CSV.

2.2.3. Requisitos de seguridad.

- Los servidores del sistema serán accedidos a través del sistema y se guardarán trazas de las acciones fundamentales en el sistema.
- Se realizarán las validaciones que garantizan el sentido de los datos con respecto al negocio, se deben discriminar en toda la entrada de caracteres especiales principalmente: comillas simples (‘ ‘), comillas dobles (“ “), paréntesis angulares (<>), asterisco (*) y cualquier otro que no tenga sentido para la lógica de la aplicación.
- Se validará la entrada de palabras reservadas SQL para mitigar algún tipo de inyección SQL.
- Integridad: Garantía de un tratamiento adecuado de las excepciones y validación de las entradas del usuario para evitar entradas inadecuadas. Validará la integridad de los datos que intercambia con las demás entidades.

2.2.4. Requisitos de Disponibilidad

- La información generada en el sistema por cada usuario, estará centralizada y en servidores seguros desde donde el sistema podrá generar diferentes reportes en dependencia de los usuarios del sistema y sus respectivos roles o niveles de acceso.
- El sistema posibilitará la gestión de aproximadamente siete mil solicitudes diarias y los usuarios de la Dirección de Gestión Administrativa del SAIME. Procesará un volumen entre dos mil y tres mil pagos por cada banco recaudador. En el momento de redactar este documento habían previsto once bancos recaudadores.

2.3. Integración con sistemas legados

Para poder realizar la integración del SIB con los sistemas de los bancos recaudadores y el Sistema de Oficinas del SAIME se establecen acuerdos con las partes implicadas para especificar las características que debe tener la comunicación en cada uno de los casos. Para más detalles consultar el anexo 4 (“Especificaciones Técnicas para la Integración con el Sistema de Oficinas del SAIME”) y el anexo 5 (“Especificaciones técnicas sobre la solución de

Software de integración entre los Bancos recaudadores y el SAIME"). Estos documentos son de uso interno para el desarrollo de la aplicación.

2.3.1. SAIME

Como parte del proceso de integración con el SAIME se necesita proveerle servicios web con el objetivo de liquidar las solicitudes. Los servicios son:

Validación de ráfaga: Valida una ráfaga que se envía desde las oficinas. Se utiliza para conocer si la ráfaga suministrada está correcta. Se considera que una ráfaga es correcta ("OK") si la información enviada por el Banco coincide con la enviada por el SAIME, en caso de no coincidir se considera que la ráfaga es incorrecta ("NoOK"), y si solo se tiene la información del SAIME se considera que la ráfaga no es totalmente válida ("semiOK"). La información del Banco siempre se considera válida. De estar correcta la ráfaga, se asocia el pago suministrado con la solicitud seleccionada.

Pagar solicitud: Se utiliza para pagar una solicitud, el SAIME envía el identificador del pago que fue dado en la validación de la ráfaga, de donde el sistema responde si puede ser efectuado el pago en dependencia del costo del trámite y de ser positivo marca el servicio como pagado, de ser negativo envía una respuesta negativa y mensaje con la causa para que el oficinista este informado, además envía el ID de la solicitud.

Cambiar estado de solicitud: Se en carga de cambiar el estado de las solicitudes enviadas según lo necesita el SAIME, recibiendo de este el número de la solicitud y el estado al cual lo necesita cambiar. Es utilizado para liquidar las solicitudes, o modificarlas porque se haya cancelado el pago.

2.3.2. Bancos

Las sucursales de cada banco se registrarán en el sistema mediante un fichero con extensión CSV que deberán proveer los bancos recaudadores donde estén registradas todas sus sucursales.

El intercambio de la información con los bancos, de: los pagos efectuados en las sucursales de los bancos recaudadores y los archivos de seguridad emitido por el sistema; se hará mediante el intercambio de ficheros XML con una estructura predefinida y convenida entre ambas partes.

El cliente accederá a la sucursal del banco de su preferencia y realizará el pago, estos se publicarán en un servidor SFTP convenido para el correspondiente banco. Mediante una VPN para garantizar la seguridad de la trasmisión de los datos un agente de actualización descargará los ficheros que se pondrán a disposición del SAIME. A este mecanismo se le agrega una validación de la integridad de los datos emitidos. En la siguiente figura se muestra el esquema descrito.

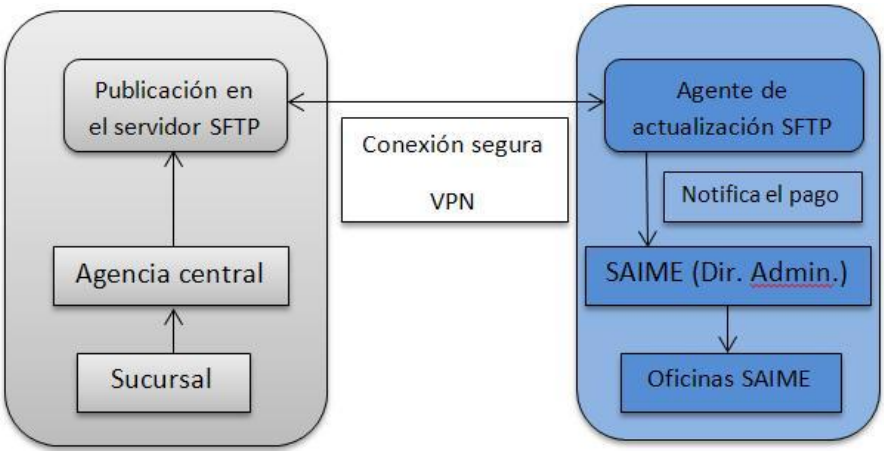


Figura 5 Esquema de trasmisión de ficheros.

La Figura 6 muestra el flujo de la validación de la ráfaga bancaria[5] solo cambian los datos esenciales que se mezclan, para mejor entendimiento de la misma auxiliarse de la siguiente leyenda:

- **C:** Cadena conformada con datos esenciales específicos del proceso.
- **CSeg:** Cadena de seguridad establecida una por cada oficina del SAIME y una por cada sucursal de los bancos recaudadores.
- **FM (C, CSeg):** Función que mezcla las cadenas para lograr como resultado una cadena V.

[5]Conjunto de datos estándar que identifican la información relevante de un pago.

- **FH64b:** Función Hash de 64 bits que se aplica sobre la cadena V y como resultado H64b, valor hash de 64 bits. La función que se propone es un CRC64.
- **FB8b (H32b):** Función que convierte el valor hash H64b a una cadena representada en el sistema numérico octal.

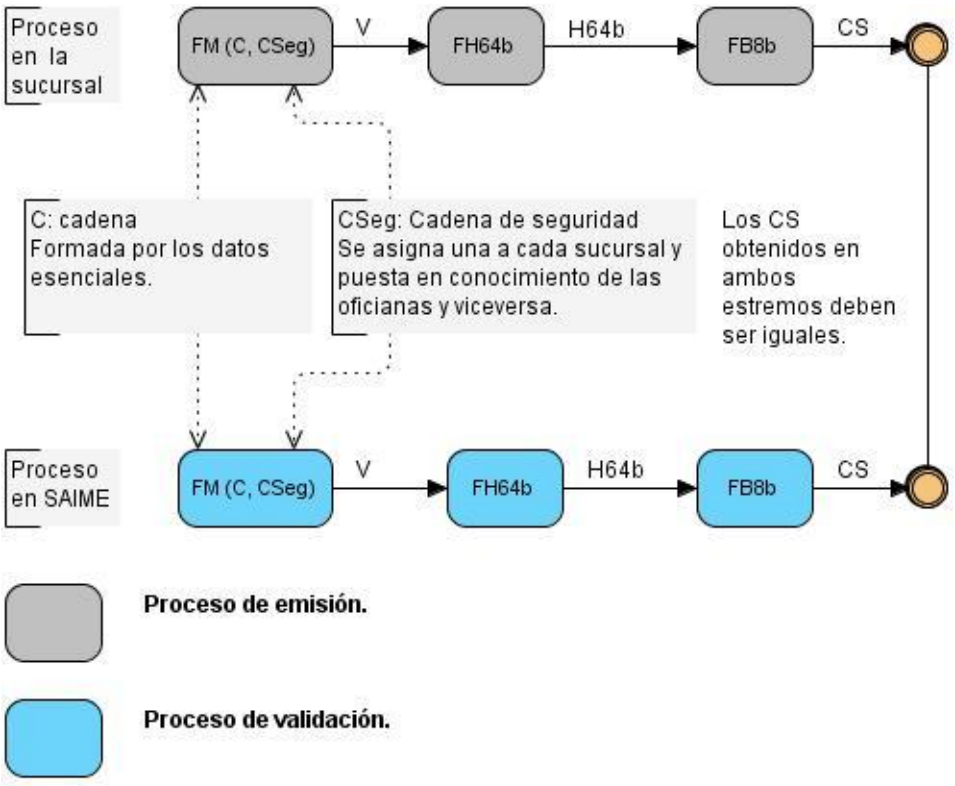


Figura 6 Flujo de la validación del voucher de una ráfaga bancaria.

La figura anterior muestra dos procesos que ocurren en paralelo, en momentos distintos, pero que deben arrojar el mismo resultado para validar la integridad de los datos.

Después del proceso se obtiene una cadena para comparar con el resultado obtenido por la otra parte.

2.4. Marco de trabajo

Las aplicaciones vistas en el epígrafe 2.1 serán desarrolladas sobre entornos de trabajos diferentes, debido a la heterogeneidad de sus lenguajes de programación y a las prestaciones que realizan. Se hace uso de *frameworks* que facilitan el desarrollo y guían el diseño de las capas lógicas.

2.4.1. Spring

En la construcción de sistemas empresariales con Spring se tiene presente conceptos tales como:

- **Inyección de Dependencia:** se delega la responsabilidad de instanciar los objetos en un archivo de configuración XML.
- **Objetos persistentes del dominio (entidades):** representa el modelo de objetos.
- **Objeto de Acceso a Datos o *Data Access Object* (DAO):** encapsulan la persistencia de los objetos de dominios.

Los DAOs con que se utilizarán en SIB, utilizarán una plantilla que agrupará de forma genérica las funcionalidades principales que se pueden presentar al trabajar con un objeto de dominio. Esta plantilla se agrupa en la librería SpringFuente-1.0 junto con ficheros de configuración de Spring, los cuales debe regir en los subsistemas que utilicen esta tecnología.

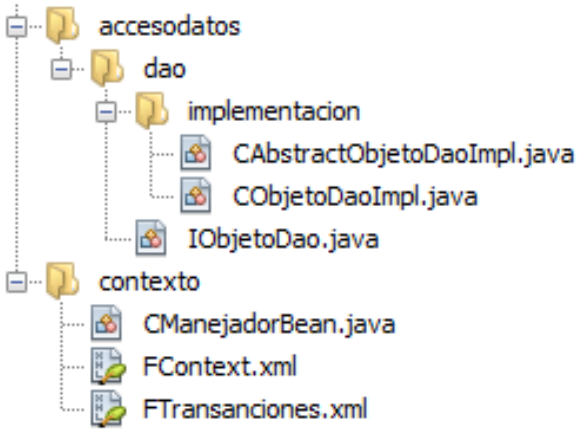


Figura 7 Librería SpringFuente-1.0

Descripción de las clases y ficheros utilizados por la librería SpringFuente-1.0:

CAbstractObjetoDaoImpl: Esta clase abstrae a los desarrolladores del trabajo con criteria, expone la mayoría de las funcionalidades que esta ofrece de manera transparente al que las utilice.

CObjetoDaoImpl: Implementa las funcionalidades básicas expuestas en IObjetoDao, y deja sentada otros métodos para localizar objetos almacenados en la base de datos, estos son particulares del *framework*, como son: ejecutar una criteria, buscar por ejemplo y ejecutar una HQL.

IObjetoDao: La interfaz expone los métodos más esenciales que deben estar presentes en cualquier DAO: métodos para persistir o salvar, eliminar y listar (por id o todos).

CManejadorBean: Se utiliza para la construcción de objetos a partir de Inyección de Dependencia. La misma carga la configuración de Spring y los ficheros donde se definen los *beans* a utilizar por la aplicación. Los *beans* son específicos de la aplicación por lo que serán definidos dentro de estas.

FContext.xml: En este fichero se define el contexto de Spring.

FTransanciones.xml: Este fichero define el ámbito de las transacciones que ocurren en el sistema, permitiendo un estándar para el trabajo con las mismas.

Las capas físicas que componen las aplicaciones empresariales para exponer sus funcionalidades varían en número de acuerdo a la arquitectura usada. La estructura del diseño de las capas lógicas del SIB es como se muestra en la siguiente figura:

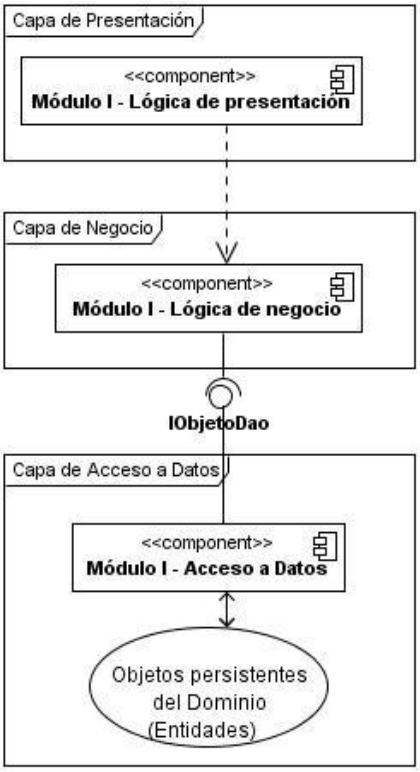


Figura 8 Capas de las aplicaciones con Spring.

Todas las capas se ejecutarán sobre las pc clientes, aprovechando los recursos de las máquinas locales.

La capa de **Acceso a Datos** mantendrá la filosofía de que es mejor programar orientado a interfaces que a clases. Las implementaciones de los DAOs estarán disponibles para los objetos de negocio haciendo uso de la inyección de dependencias, configurada en el contenedor de inversión de control de Spring Framework.

En la capa de **Negocio** radicarán los objetos de negocio encargados de separar los datos y la lógica de negocio usando el modelo de objetos, permitiendo usar los objetos de dominio solo como transferencia de objetos, que se mueven entre las capas arquitectónicas de la aplicación. Las políticas transaccionales de la aplicación serán planteadas al nivel de los objetos de negocio. Los componentes de negocio brindarán clases a ser usadas por la capa de presentación.

La capa de **Presentación** acogerá todo lo referente a aspectos de presentación.

2.4.2. Symfony

Este *framework* automatiza los patrones utilizados para resolver las tareas más comunes, le proporciona estructura al código fuente, forzando al desarrollador a crear códigos más legibles y más fáciles de mantener.

Symfony toma lo mejor del estilo arquitectónico MVC y lo implementa de forma tal que el desarrollo de aplicaciones se hace rápido y sencillo. Define un controlador frontal y el *layout* comunes para todas las acciones de la aplicación. El controlador frontal es un componente que solo tiene código relativo al MVC, por lo que no es necesario crear uno, ya que este *framework* lo genera de forma automática.

Las clases de la capa del modelo también se generan automáticamente, en función de la estructura de datos de la aplicación. La librería Doctrine se encarga de esta generación automática, ya que crea el "esqueleto" o estructura básica de las clases y genera automáticamente el código necesario. La abstracción de la base de datos es completamente transparente para el programador, ya que se realiza de forma nativa mediante PDO (*PHP Data Objects*). Así, si se cambia el sistema gestor de bases de datos en cualquier momento, no se debe reescribir ni una línea de código, ya que tan solo es necesario modificar un parámetro en un archivo de configuración.

Por último, la lógica de la vista se puede transformar en un archivo de configuración sencillo, sin necesidad de programarla.

Todo el flujo de trabajo descrito anteriormente para este framework se muestra en la Figura 9.

Dentro de un proyecto, las operaciones se agrupan de forma lógica en aplicaciones. Cada aplicación está formada por uno o más módulos, la estructura de cada aplicación es como se detalla en la Tabla 1. Un módulo normalmente representa a una página web o a un grupo de páginas con un propósito relacionado. Los módulos almacenan las acciones, que representan cada una de las operaciones que se pueden realizar en los mismos, en la tabla 2 se muestra la estructura de directorio típica de un módulo.

Symfony proporciona una estructura en forma de árbol de archivos para organizar de forma lógica el contenido, además de ser consistente con la arquitectura MVC utilizada y con la

agrupación proyecto / aplicación / módulo. En la Tabla 3 se detalla la estructura de los directorios en la raíz de los proyectos Symfony.

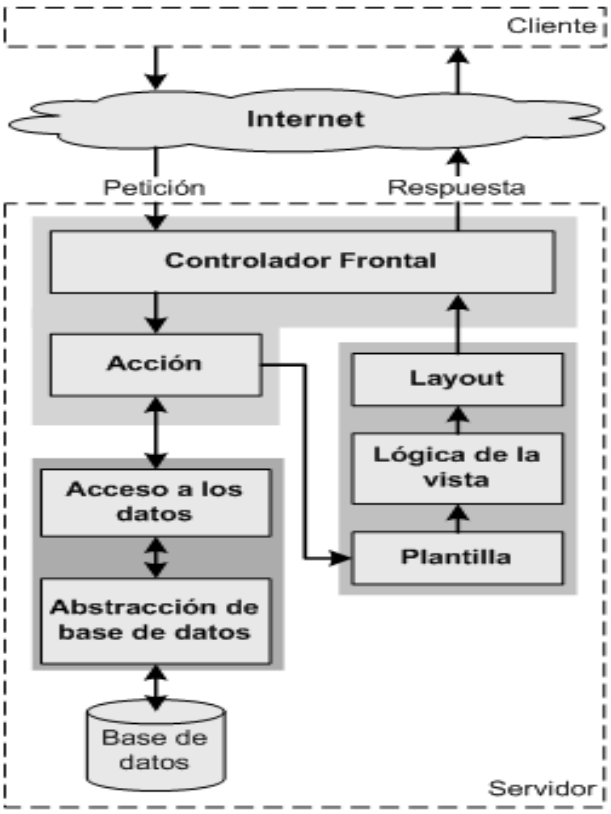


Figura 9 Flujo de trabajo de Symfony.

Tabla 1 Estructura de directorio de cada aplicación.

Directorio	Descripción
config/	Contiene muchos archivos de configuración creados con YAML. Aquí se almacena la mayor parte de la configuración de la aplicación, salvo los parámetros propios del *framework*. También es posible redefinir en este directorio los parámetros por defecto si es necesario.
i18n/	Contiene todos los archivos utilizados para la internacionalización de la aplicación, sobre todo los archivos que traducen la interfaz. La internacionalización también se puede realizar con una base de datos, en cuyo caso este directorio no se utilizaría.
lib/	Contiene las clases y librerías utilizadas exclusivamente por la aplicación.

modules/	Almacena los módulos que definen las características de la aplicación.
templates/	Contiene las plantillas globales de la aplicación, es decir, las que utilizan todos los módulos. Por defecto contiene un archivo llamado *layout.php*, que es el *layout* principal con el que se muestran las plantillas de los módulos.

Tabla 2 Estructura de directorio típica de un módulo.

Directorio	Descripción
actions/	Normalmente contiene un único archivo llamado *actions.class.php* y que corresponde a la clase que almacena todas las acciones del módulo. También es posible crear un archivo diferente para cada acción del módulo.
config/	Puede contener archivos de configuración adicionales con parámetros exclusivos del módulo.
lib/	Almacena las clases y librerías utilizadas exclusivamente por el módulo.
templates/	Contiene las plantillas correspondientes a las acciones del módulo. Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada *indexSuccess.php*

Tabla 3 Estructura de los directorios en la raíz de los proyectos Symfony.

Directorio	Descripción
apps/	Contiene un directorio por cada aplicación del proyecto (normalmente, *frontend* y *backend* para la parte pública y la parte de gestión respectivamente).
cache/	Contiene la versión cacheada de la configuración y (si está activada) la versión cacheada de las acciones y plantillas del proyecto. El mecanismo de caché utiliza los archivos de este directorio para acelerar la respuesta a las peticiones web. Cada aplicación contiene un subdirectorio que guarda todos los archivos PHP y HTML preprocesados.
config/	Almacena la configuración general del proyecto.
data/	En este directorio se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de bases de datos de *SQLite*.

doc/	Contiene la documentación del proyecto, formada por tus propios documentos y por la documentación generada por *PHPdoc.*
lib/	Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio model/ guarda el modelo de objetos del proyecto.
log/	Guarda todos los archivos de log generados por *Symfony*. También se puede utilizar para guardar los logs del servidor web, de la base de datos o de cualquier otro componente del proyecto. *Symfony* crea un archivo de log por cada aplicación y por cada entorno.
plugins/	Almacena los *plugins* instalados en la aplicación.
test/	Contiene las pruebas unitarias y funcionales escritas en PHP y compatibles con el *framework* de pruebas de *Symfony*. Cuando se crea un proyecto, *Symfony* crea algunos pruebas básicas
web/	La raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se encuentran en este directorio

2.5. Vista de Casos de Uso

Según RUP en la Vista de Casos de Usos de la Arquitectura se relacionan aquellos que son arquitectónicamente significativos, los cuales contienen las funcionalidades críticas para el sistema. Se refieren como críticos cuando son: funciones de mayor importancia o la razón de existir del sistema, de igual forma si son los de mayor frecuencia de uso, presentan riesgo técnico que debe ser mitigado o que sirven para validar la arquitectura propuesta por el uso de la mayoría de los elementos de la misma.

Teniendo en cuenta los objetivos de cada módulo se realiza la distribución de los casos de usos de la manera siguiente:

Generar Código Seguridad:

Configurar conexión de servidores: Permite crear la conexión a los servidores FTP ubicados en los bancos y probar la misma.

Generar archivo de seguridad: Se generan los archivos de seguridad a partir de una fecha de comienzo de uso de las cadenas de seguridad, después se envían y notifican los ficheros a los bancos.´

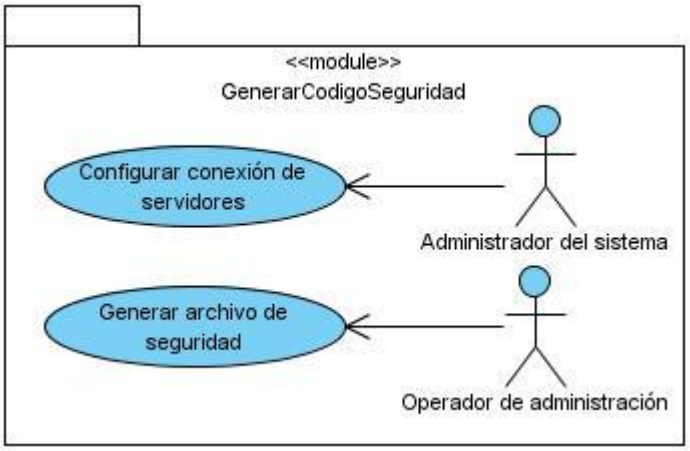


Figura 10 Vista de CU: Módulo Generar Código Seguridad

Lectura y Común:

Administrar lecturas de pago: Gestiona las lecturas de los ficheros.

Ejecutar lecturas automáticas: Permite iniciar la ejecución de las lecturas de los ficheros de forma automática, la misma puede ser detenida, reiniciada y cerrada.

Validar CSS: Se toman los datos de la ráfaga bancaria y se valida el CSS.

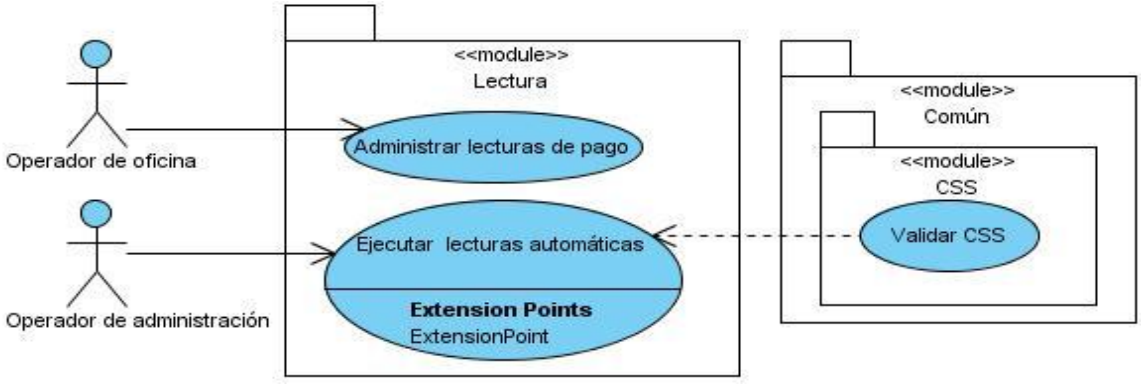


Figura 11 Vista de CU: Módulo Lectura y Común

Módulo Pago:

Registra pago ordinario: Guarda los pagos de los clientes y los asocia con las solicitudes correspondientes.

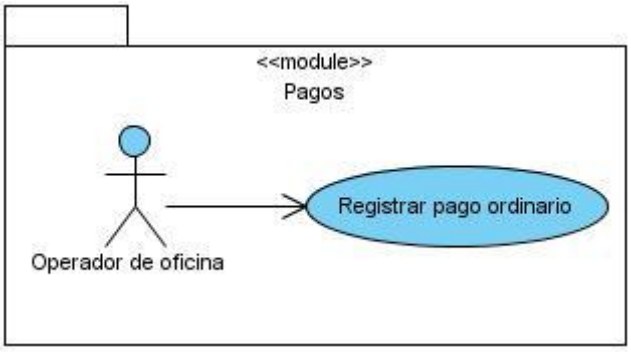


Figura 12 Vista de CU: Módulo Pago

La aplicación Solicitud de pago de servicio, está dividido en dos aplicaciones. Se presentarán los CU por módulos, de aquellos que contienen CU arquitectónicamente significativos.

Módulo Solicitudes:

Gestionar Solicitud: Los clientes acceden al sitio con la finalidad de realizar una solicitud con los servicios de su interés, esta se puede modificar o eliminar.

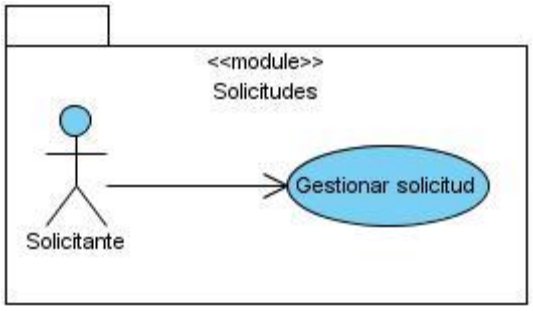


Figura 13 Vista de CU: Módulo Solicitudes

Módulo Administración:

Administrar servicios: Gestiona los servicios que estará prestando el proveedor.

Administrar bancos recaudadores: Gestiona los bancos recaudadores que tienen contrato con el proveedor.

Administrar cuentas recaudadoras: Gestiona las cuentas recaudadoras del proveedor con sus bancos recaudadores.

Administrar lista de distribución: Gestiona las listas de distribución que se usarán para enviar los avisos promocionales a los clientes.

Figura 14 Vista de CU: Módulo Administración

2.6. Vista de la implementación

Esta vista comprende a grandes rasgos todos aquellos artefactos que se utilizan para ensamblar el sistema y ponerlo en producción, ya listo para su distribución física. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes.

En la Figura 15 se muestra el diagrama de implementación de las aplicaciones de escritorio. Está compuesto por todos los componentes reutilizables de los cuales dependen las aplicaciones principales, los mismos son:

Común.jar: Contiene módulos comunes para las aplicaciones.

Correo.jar: Componente para enviar correo electrónicos.

Codificación.jar: Contiene todo lo referente a los algoritmos de codificación y sus políticas de uso.

SFTP.jar: Permite las conexiones con servidores SFTP.

SpringFuente.jar: Facilita el trabajo con Spring.

Fichero.jar: Facilita el trabajo con los principales ficheros que se utilizarán en el sistema.

MonitorWeb.aar: Proveerá servicios web para monitorear la ejecución de las lecturas automáticas.

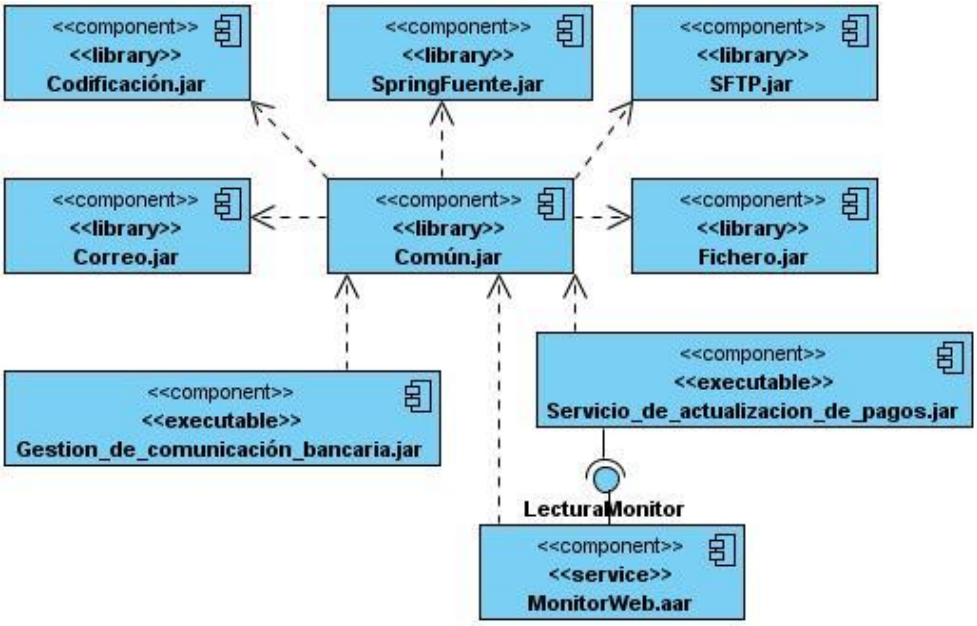


Figura 15 Vista de los componentes de las aplicaciones de escritorio.

Los diagrama de implementación de las aplicaciones web son más ricos en información por lo que se dividirá por módulos su representación. Para ver dichos diagramas consultar los Anexos 1, 2 y 3.

La capa del modelo es generada por Doctrine, *framework* ORM integrado con *Symfony* para el desarrollo de la aplicación *web.* El modelo está formado por un conjunto de clase por cada tabla de la base de datos; las que le antecede la palabra “Base” son controladas por el *framework*, las que le precede la palabra “Table” son para realizar consultas que devuelvan lista de objetos, y la que tiene el nombre referente a la tabla en la base de datos son para realizar operaciones sobre un objeto de la tabla. *Symfony* se encarga de generar y controlar las vistas y los controladores. Cada módulo está compuesto por un conjunto de plantillas que se

cargan en las vistas principales que se desean mostrar en los formularios. Como parte del trabajo de aplicaciones *web* y del *framework* en común vienen asociados: ficheros css, librerías para el trabajo con *javascript* y ficheros de configuración del *framework.*

2.7. Vista de despliegue

Esta vista muestra los nodos físicos que abarca el SIB, en la figura siguiente se muestra el diagrama de la vista de despliegue.

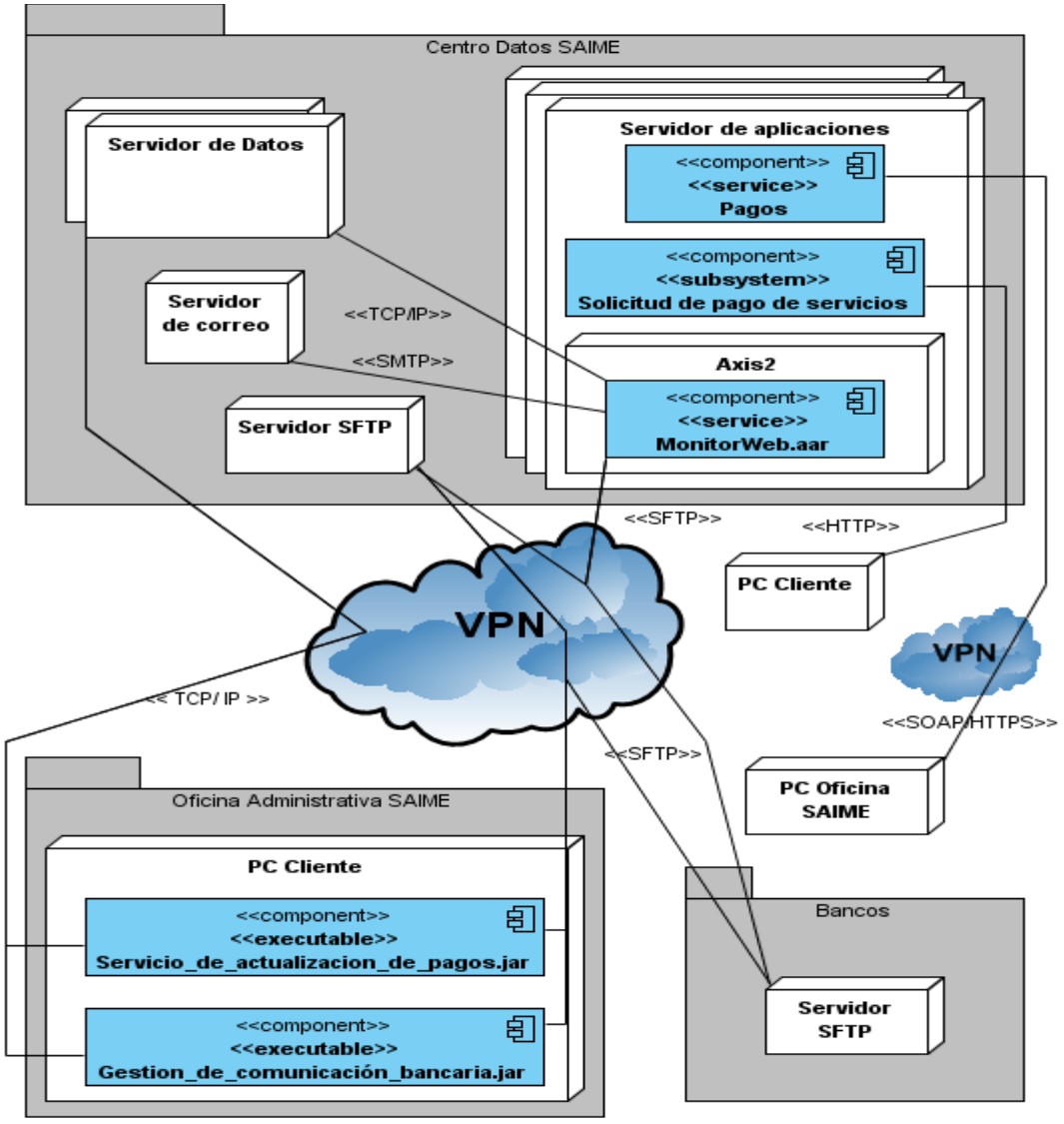


Figura 16 Diagrama de despliegue.

El despliegue del sistema estará ubicado en dos áreas principales, el Centro Datos SAIME y la oficina administrativa del mismo. En la primera se encuentra un servidor web en el cual se desplegará la aplicación web Solicitud de pago de servicio, que estará de cara a Internet, para permitir el acceso de los clientes a los servicios que necesiten para el pago de sus trámites. En el mismo servidor web estarán además las aplicaciones Pagos y MonitorWeb. En el caso de la primera ofrece un grupo de servicios web al Sistema de Oficinas del SAIME para registrar los pagos de los clientes; la segunda se ejecutará sobre Axis2 y ofrecerá servicios web para la aplicación Actualización de Pagos. En las oficinas del SAIME se desplegarán las aplicaciones Gestión de Comunicación Bancaria y Actualización de Pagos. Estas áreas deben estar ubicadas en la misma VPN que se pacte con los bancos para permitir la comunicación con los servidores SFTP; de igual manera los servicios web consumidos desde el Sistema de Oficinas del SAIME se consumirán sobre una VPN.

Existirá comunicación con los bancos a través de servidores SFTP donde ambos sistemas (Sistema en Banco (SEB) y SIB) colocarán y leerán archivos XML indistintamente. Debido a restricciones en algunos bancos estos servidores estarán ubicados en el Centro Datos SAIME o en los bancos dependencia de cada caso.

Servidor de correo: Servidor de correo disponible para el envío de los correos que demanden la aplicación.

Servidor de aplicaciones: Se hospeda la aplicación web Solicitud de pago de servicio, Pagos y el componente auxiliar MonitorWeb.aar corriendo sobre Axis2. Teniendo en cuenta el nivel de disponibilidad para el sistema, se propone utilizar tres servidores de aplicaciones, donde uno realice la tarea de balanceador de carga, para disminuir la probabilidad de que se caiga el servidor por un exceso de peticiones.

PC Cliente: El nodo ubicado en SAIME se corresponde con la máquina donde se encuentran instalados las dos aplicaciones de escritorio. El otro nodo representa el acceso por Internet, tanto de los clientes como de los trabajadores del SAIME con fines administrativos.

PC Oficina SAIME: Representa las aplicación del Sistema de Oficinas del SAIME.

Gestor de Base de Datos PostgreSQL: Representa el servidor donde se hospeda la base de datos a la que accederá cada aplicación. Se propone utilizar un cluster de servidores formado por dos nodos.

Servidor SFTP: Estos nodos representan la ubicación de cada servidor FTP, para intercambiar ficheros con los bancos. Cada banco deberá proveer el suyo, en caso contrario en el Centro Datos se ubicará otro para prestar servicios aquellos bancos que no dispongan de servidor.

2.8. Conclusiones

La propuesta de arquitectura base para el SIB permite el diseño basado en capas lógicas con el uso de *framework para* conseguir un mantenimiento más sencillo de las aplicaciones y simplificar el desarrollo.

Con la modelación de cada una de las vistas se brinda la posibilidad de definir la estructura de los subsistemas, así como sus relaciones y comportamiento con otros componentes.

El ambiente de desarrollo propuesto permite guiar el mismo, acota las funcionalidades de cada componente, y define cómo se debe estructurar y configurar cada subsistema.

CAPÍTULO 3: EVALUACIÓN DE LA PROPUESTA DE SOLUCIÓN

Introducción

Las decisiones de diseño que hacen patente la arquitectura son precisamente las que más van a condicionar el desarrollo del sistema, las de más alcance, y por tanto, las más difíciles de corregir en caso de error. La detección precoz de errores es una de las principales tareas de la informática. El coste de corregir un error se hace tanto mayor cuanto más tarde se detecta en el ciclo de vida de un proyecto. Por ello es necesario disponer de métodos fiables para evaluar la arquitectura y aplicarlos tan pronto como sea posible.

3.1. Evaluación de las arquitecturas

Al definir una arquitectura hay una pregunta que siempre está latente y es ¿cómo estar seguro que la elegida es la correcta para el sistema informático que se necesita desarrollar?

No es una pregunta fácil de responder ya que la arquitectura es la piedra fundamental de cualquier software y define además los atributos de calidad de un sistema.

Las decisiones arquitectónicas ocurren en las primeras etapas del diseño y afectan a la estructura misma del sistema, condicionando todo el posterior proceso de desarrollo. Es evidente que las arquitecturas deben evaluarse, confrontándolas con los atributos de calidad que dependen de ellas.

Un atributo de calidad es una característica que afecta a un elemento. Donde el término “característica” se refiere a aspectos no funcionales y el término “elemento” a componente.

Ventajas de evaluar la arquitectura:

- Cuanto más temprano se encuentre un problema en un proyecto de software, mejor.
- Realizar una evaluación de la arquitectura es la manera más económica de evitar desastres.
- Analizar y evaluar la calidad exigida por los usuarios.
- Decisiones de diseño

- Restricciones de Implementación.
- Fija la estructura organizacional, tanto del desarrollo, construcción y ejecución del sistema.
- Logra los atributos de calidad.
- Permite el prototipado.
- Permite estimaciones más certeras.

- Abstracción transferible entre sistemas.

La evaluación de la arquitectura debe arrojar como resultados una lista de posibles riesgos o no riesgos, a tener en cuenta durante el desarrollo del sistema informático.

3.2. Métodos para evaluar la arquitectura

3.2.1. SAAM

SAAM (*Architecture Analysis Method*): no evalúa la arquitectura respecto de atributos de calidad abstractos, sino respecto de requisitos concretos. La evaluación se realiza en un contexto muy específico, asumiendo que los atributos de calidad no existen como conceptos aislados, sino que tienen sentido dentro de un contexto. El método debe consumir pocos recursos y debe poder aplicarse antes de que el sistema este implementado, aun cuando no se conozcan todos los detalles del sistema final. La aplicación del método requiere de una descripción clara, pero en general no muy detallada de la arquitectura. SAAM es un método conducido por escenarios. Si la calidad de la arquitectura depende del contexto en el que se evalúa, es necesario definir de alguna manera ese contexto. El método SAAM se desarrolla en seis etapas:

1. Desarrollo de escenarios.
2. Descripción de las arquitecturas candidatas.
3. Clasificación de escenarios.
4. Realización de escenarios.

5. Interacción entre escenarios.
6. Evaluación global.

3.2.2. ADR

ADR (*Active Design Review*): es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto (2).

3.2.3. ARID

ARID (*Active Reviews for Intermediate Design*): es un método de bajo costo y gran beneficio, es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. ARID es un híbrido entre ADR y ATAM (8). Se basa en ensamblar el diseño de los *stakeholders* para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los *stakeholders*. Este método consta de 9 pasos agrupados en dos fases (Actividades Previas y Evaluación).

3.2.4. ATAM

ATAM (*Architecture Trade-off Analysis Method*): está inspirado en tres áreas distintas, los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros.

El ATAM puede usarse para crear definiciones preliminares de la arquitectura, analizar decisiones arquitectónicas en ausencia de código, analizar diversas arquitecturas candidatas para el desarrollo de un sistema y analizar sistemas ya existentes. Su propósito principal es:

descubrir las consecuencias de las decisiones arquitectónicas a la luz de los requisitos de los atributos de calidad. (21)

Asumiendo que:

Los análisis de atributos de calidad específicos deben realizarse teniendo en cuenta su dependencia con el resto de los atributos del sistema. Los atributos de calidad no son independientes, sino que interaccionan entre sí a través de las relaciones estructurales impuestas por la arquitectura.

Las ideas básicas en las que se sustenta el método son las siguientes:

- Es un método conducido por escenarios.
- No evalúa la arquitectura respecto a atributos de calidad abstractos, sino respecto a requisitos concretos.
- Requiere de una descripción de las estructuras del sistema que se pretenden evaluar, mientras mayor sea la influencia en los atributos de calidad más detallada debe ser la descripción.
- Su ejecución debe consumir pocos recursos y realizarse en un lapso de tiempo relativamente corto.
- Toma en consideración tanto los requisitos técnicos, como aspectos sociales y de negocio.
- La caracterización de los atributos de calidad.
- La noción de estilo arquitectónico.

El proceso de evaluación permitirá descubrir los **riesgos**, **puntos sensibles** y **puntos de compromiso** asociados a las decisiones arquitectónicos:

- Los **riesgos** se producen cuando hay que elegir entre varias decisiones de diseño arquitectónico o cuando, una vez tomada dicha decisión, no pueden determinarse claramente los efectos que produce sobre alguno de los atributos de calidad. También

hay riesgos asociados a la gestión del proyecto y a la comunicación entre las partes implicadas.

- Los **puntos sensibles** de una arquitectura son aquellas propiedades de sus componentes y relaciones altamente correlacionadas con el cumplimiento de un determinado atributo de calidad y que por tanto son críticas para el cumplimiento del mismo.
- Los **puntos de compromiso** son los puntos sensibles que involucran a más de un atributo de calidad. Los puntos de compromiso son los lugares donde se producen los conflictos entre atributos, o dicho de otra manera, los lugares en los cuales los atributos interaccionan y no pueden considerarse por separado.

Con el objeto de identificar los riesgos, puntos sensibles y puntos de compromisos, ATAM utiliza tres elementos: los escenarios de alta prioridad, las cuestiones específicas de atributo y los estilos arquitectónicos.

Durante la evaluación, el arquitecto realiza los escenarios a través de las estructuras definidas en la arquitectura. Para ello deben identificarse los componentes y conectores involucrados en su realización y plantearse las cuestiones necesarias para identificar los riesgos, los puntos sensibles y los puntos de compromiso.

El propósito del método no es producir análisis precisos de los atributos de calidad, sino identificar tendencias que permitan determinar si el enfoque arquitectónico adoptado es el correcto. El objetivo no es caracterizar de forma precisa el comportamiento de los atributos, sino descubrir en qué lugares y de qué manera les afectan las decisiones de diseño arquitectónicos. Una vez encontrados los puntos sensibles y de compromiso se puede determinar la necesidad de modelar con mayor exactitud los componentes y las relaciones involucrados en los mismos y analizar su comportamiento en relación con los atributos de calidad. Los resultados de dichos análisis pueden incorporarse al método, pero los análisis en sí están fuera del mismo.

La clasificación de los escenarios es coherente con la clasificación de los atributos de calidad. Todas las clasificaciones son discutibles, sin embargo para la evaluación lo realmente

importante es el consenso acerca de los escenarios que han de considerarse y no tanto su clasificación, que puede realizarse según diferentes taxonomías.

El método ATAM consta de 9 pasos. Aunque dichos pasos están numerados y sugieren un desarrollo lineal del método, su orden puede variar de acuerdo con las necesidades del proyecto y puede ser necesario realizar varias iteraciones de los mismos. La importancia relativa de cada paso y el tiempo que puede consumir dependen de las características de cada proyecto. Los pasos son los siguientes:

1. **Presentación del método:** Se explica el método a todas las partes implicadas, sus fundamentos y pasos, las técnicas para la obtención y refinado de escenarios, el significado de los riesgos, los puntos sensibles y los puntos de compromiso, etc. El objetivo es que todas las partes implicadas entiendan el proceso y se sientan involucradas en él.
2. **Descripción de los factores de negocio**: El sistema en cuyo desarrollo se va a utilizar la arquitectura debe ser perfectamente entendido por todas las partes. El cliente describe el sistema desde la perspectiva de negocio.
3. **Presentación de la arquitectura:** El equipo de desarrollo presenta la arquitectura describiendo el enfoque adoptado para el logro de los atributos de calidad definidos en el paso anterior. Esta descripción debe realizarse en un nivel de abstracción inteligible para todas las partes implicadas, pero en todo caso suficiente para determinar la conformidad de la arquitectura con los atributos que se pretende evaluar.
4. **Identificación de estilos arquitectónicos:** Se identifican claramente los estilos arquitectónicos usados en la arquitectura. Dichos estilos representan los medios mediante los cuales la arquitectura puede alcanzar sus atributos de calidad y definen la forma en que un sistema puede crecer, modificarse, integrarse con otros sistemas y responder a estímulos externos.
5. **Generación del árbol de utilidad:** A partir de los factores de negocio expuestos por el cliente (paso 2), se genera un árbol de utilidad que traduzca sus expectativas a requisitos específicos de los atributos de calidad. Se describen escenarios en términos

de los estímulos que recibirá el sistema y de las respuestas esperadas y se priorizan en función, tanto de su importancia para el éxito del proyecto como de los riesgos que implican. El árbol de utilidad sirve de guía para los siguientes pasos del método, indicando las áreas hacia las que el evaluador debe dirigir sus esfuerzos. Cada una de las hojas del árbol de utilidad define por sí misma un escenario.

6. **Análisis de los estilos arquitectónicos:** Una vez que el alcance de la evaluación ha sido definido mediante el árbol de utilidad (paso 5) y los estilos arquitectónicos han sido identificados (paso 4), el equipo de evaluación puede empezar a analizar la arquitectura. El primer paso consiste en asociar los escenarios más prioritarios del árbol de utilidad con los estilos arquitectónicos empleados en la arquitectura. Al final de este paso, el equipo evaluador debe tener ya una idea general de los aspectos más importantes de la arquitectura. A partir de aquí el objetivo es refinar este conocimiento y llegar a conclusiones acerca de la adecuación de la arquitectura con sus objetivos.
7. **Generación de escenarios y asignación de prioridades a los mismos:** En este paso el objetivo es generar escenarios y asignarles diferentes prioridades. La priorización de escenarios se realiza mediante votación de todas las partes implicadas. Si el tiempo y los recursos disponibles para la evaluación son limitados el equipo de evaluación puede considerar solo los escenarios más prioritarios. Una vez definidos y priorizados los escenarios se contrastan con las hojas del árbol de utilidad. Cualquier discrepancia entre ambos conjuntos de escenarios supone una discrepancia entre los puntos de vista del personal técnico que ha elaborado el árbol de utilidad y el resto de las partes implicadas.
8. **Análisis de los estilos arquitectónicos:** Este paso es una repetición del paso 6, pero tomando en consideración los nuevos escenarios definidos en el paso 7. Si el árbol de utilidad ha sido bien definido este paso debe ser muy breve, pues los escenarios definidos por las hojas del árbol de utilidad deben ser básicamente los mismos que los definidos en el paso 7. Si no es así, debe volverse al paso 4, redefinir el árbol de utilidad y repetir los pasos 5, 6 y 7.

9. **Presentación de resultados:** Este es el último paso y es donde la información generada durante la ejecución del método se organiza, resume y presenta a todas las partes implicadas. Esta presentación debe incluir:

- Documentación de los estilos arquitectónicos.
- Conjunto de escenarios y su orden de prioridad.
- El árbol de utilidad.
- Los riesgos y no-riesgos descubiertos.
- Los puntos sensibles y los puntos de compromiso.

3.3. Aplicación del método ATAM a la arquitectura propuesta

Existen varios métodos de evaluación de arquitectura algunos más completos que otros y no todos abarcan la misma área. El objetivo perseguido para evaluar la arquitectura propuesta es validar la relación de los requerimientos, atributos de calidad y restricciones arquitectónicas para asegurar que el sistema a ser construido cumple con las necesidades de los *stakeholders*. La evaluación de una arquitectura pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Es por ello que el método utilizado es el ATAM, se cree que los demás métodos no se ajustan a los objetivos perseguidos, además este método evalúa con más profundidad, en relación con otros métodos, cuestiones referentes a la arquitectura, como son: los atributos de calidad.

Algunos de sus pasos ya han sido ejecutados en los capítulos anteriores y otros han debido adaptarse a las circunstancias en las que se ha desarrollado este trabajo de diploma. Por ello, es necesario comentar brevemente como se ha ejecutado el método.

1. Presentación del método: En el epígrafe anterior se realizó el estudio de los puntos fundamentales correspondiente a este método.
2. La descripción de los factores de negocio: En la introducción del presente trabajo de diploma se realiza un análisis sobre la situación problémica que afronta el SAIME en el proceso de recaudación de los pagos bancarios, de la cual se deduce la carencia de

integración de esta institución con sus bancos recaudadores; de igual forma la ausencia de mecanismos de seguridad y validación de los pagos de los trámites efectuados por sus clientes. De esta situación surge la necesidad de proveer un mecanismo de comunicación segura entre el SAIME y los bancos recaudadores. En la actualidad las empresas tienden a realizar pagos en línea, vía web. A pesar de esto en los requisitos firmados no se concibe un sistema con estas características, pero no por ello se deja de tener en cuenta como futuras modificaciones del sistema. El equipo de desarrollo, para dar solución a esta situación, cuenta con un corto período de tiempo y un bajo presupuesto; de ahí que la solución esté enfocada al uso de herramientas libres, que recorten los precios por cuestión de pagos de licencias.

3, 4. Descripción de la arquitectura, Identificación de estilos arquitectónicos: En el Capítulo 2 se realiza la descripción de la arquitectura propuesta, identificando los estilos y patrones que la especifican.

5. La generación del árbol de utilidad: Todas las clasificaciones de los escenarios son discutibles, sin embargo, para la evaluación lo realmente importante es el consenso acerca de los escenarios que han de considerarse y no tanto su clasificación, que puede realizarse según diferentes criterios. En las tablas que a continuación se detallan se especifican los aspectos a evaluar para los atributos de calidad: Modificabilidad, Seguridad y Disponibilidad, de estos aspectos se analizan los escenarios, las respuestas esperadas para los mismos y su prioridad.

Tabla 4 Árbol de utilidad: Modificabilidad.

Árbol de Utilidad: Modificabilidad			
Aspectos	**Escenarios**	**Respuestas Esperadas**	**Prioridad**
Portabilidad	P1: Cambio de plataforma.	Ningún efecto.	Bajo
	P2: Reemplazamiento de una unidad física.	Compilar el nodo específico o alguna librería relacionada con el nodo en concreto.	Bajo
Adaptabilidad	A1: Añadir una nueva funcionalidad.	La complejidad de los cambios debe ser pequeña y su número limitado.	Medio
	A2: Cambio de tecnología de	El cambio se concentra en los componentes relacionados.	Alto

	desarrollo.		
	A3: Nueva forma de pago	La complejidad de los cambios debe ser pequeña y su número limitado.	Medio
Compatibilidad	CP1: Cambio en la comunicación con sistemas externos.	Los cambios concentran en los mecanismos de comunicación.	Medio
	CP2: Cambio de tecnología en los sistemas externos.	Ningún efecto.	Bajo

Tabla 5 Árbol de utilidad: Seguridad.

Árbol de Utilidad: Seguridad			
Aspecto	**Escenarios**	**Respuestas Esperadas**	**Prioridad**
Confidencialidad	C1: Acceso no autorizado.	Detección. La aplicación requiere autenticación en todos sus nodos.	Alto
	C2: Alteración de la integridad de la información.	Detección.	Alto
	C3: Solicitudes con identidades falsas.	Detección.	Bajo
Seguridad interna.	SI1: Alteraciones en los datos de los ficheros.	Posible detección. Depende de si se filtra el proceso de codificación.	Alto
	SI2: Ocurrencia de acciones anómalas.	Auditoría	Medio
Seguridad externa	SE1: Falsificación del voucher bancario.	Detección.	Alto
	SE2: Acceso a la información que viaja por la red.	Ningún efecto.	Medio

Tabla 6 Árbol de utilidad: Disponibilidad.

Árbol de Utilidad: Disponibilidad			
Aspecto	**Escenarios**	**Respuestas Esperadas**	**Prioridad**
Ocurrencia de fallo	F1: Deja de funcionar un servidor.	El fallo se detecta y de manera automática levanta el servidor de respaldo.	Medio
	F2: No hay conexión con los servidores de aplicaciones.	Se prestaran los servicios que prescindan de los mismos.	Bajo
	F3: Fallo en la conexión con la base de datos.	Detección y control de las posibles excepciones.	Bajo
Latencia de datos	L1: Ataques de	Denegar el acceso	Alto

	denegación de servicio		
	L2: Alta concurrencia	Balancear la carga de las peticiones.	Alto

6, 7. Generación de escenarios.

Atributo de calidad: **Modificabilidad.**

Escenarios de portabilidad: P1, P2.

- P1: Cambio de plataforma.
- P2: Reemplazamiento de una unidad física.

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios P1 y P2:

- Utilización de tecnologías multiplataforma.
- Desarrollo de componentes reutilizables.
- Agrupar funcionalidades comunes a cada subsistema.

Realización de los escenarios P1 y P2:

Satisfactoria, los cambios se realizaron al nivel requerido, un cambio puede ir desde remplazar una biblioteca que utiliza un subsistema hasta reemplazar versiones de subsistemas, los que no alteran el funcionamiento de los demás subsistemas.

Escenarios de Adaptabilidad: A1..A3.

- A1: Añadir una nueva funcionalidad.
- A2: Cambio de tecnología de desarrollo.
- A3: Nueva forma de pago

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios A1..A:

- Uso de estilos arquitectónicos basados en capas lógicas.
- Uso de soluciones genéricas.
- Estandarización de la comunicación.

- Uso de operaciones con funcionalidades conceptuales.

Realización de los escenarios A1...A3:

Satisfactoria, los cambios están identificados y tienen un número limitado. En el caso de una nueva funcionalidad el número de cambios varía en dependencia de qué tanto tienen que ver estas funcionalidades con el negocio inicialmente descrito. La solución de A2es la comentada en los escenarios de portabilidad.

Escenarios de Compatibilidad: CP1, CP2.

- CP1: Cambio en la comunicación con sistemas externos.
- CP2: Cambio de tecnología en los sistemas externos.

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios CP1, CP2:

- Estandarización de la comunicación con los bancos recaudadores mediante el uso de ficheros XML.
- Publicación de servicios web para el Sistema de Oficinas del SAIME

Realización de los escenarios:

Satisfactoria, los cambios se centran fundamentalmente en las interfaces y mecanismos convenidos por las partes implicadas y su expansión en el resto de la aplicación es pequeña o nula. Los cambios de tecnología en sistemas externos tienen poco impacto en la aplicación.

Atributo de calidad: **Seguridad.**

Escenarios de Confidencialidad: C1...C3.

- C1: Acceso no autorizado.
- C2: Alteración de la integridad de la información.
- C3: Solicitudes con entidades falsas.

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios C1...C3:

- Suma de control del contenido (sha1).

- Algoritmos de codificación
- Procedimientos internos pactados con los clientes y asociados (Figura 6).
- Autenticación.

Realización de los escenarios:

Medianamente satisfactoria, los algoritmos utilizados son referenciados en el documento de especificaciones técnicas. (22)Para el control del acceso de los clientes se verificará contra la información almacenada en la base de datos importada de la aplicación del SAIME. Un documento de identificación de un ciudadano puede ser utilizado por otro y de esta forma estar en presencia de una suplantación de identidad, ahí pasaría a jugar un papel fundamental las verificaciones que realiza el personal calificado para esta tarea en el SAIME, al estar en presencia del factor humano se considera que este proceso no es del todo óptimo.

Escenarios de Seguridad interna: SI1, SI2.

- SI1: Alteraciones en los datos de los ficheros.
- SI2: Ocurrencia de acciones anómalas.

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios SI1, SI2:

- Registro de operaciones significativas del sistema.
- Suma de control del contenido (sha1).
- Algoritmos de codificación.
- Procedimientos internos pactados con los clientes y asociados (Figura 6).

Realización de los escenarios:

Satisfactoria, en el sistema se registran los archivos *logs* que contienen el flujo de dato, modificaciones, los usuarios que las realizaron, la fecha/hora y otros datos de interés, de esta forma quedarían registradas todas las acciones que se realizan en la aplicación y se posibilita la detección de sucesos anómalos.

Escenarios de Seguridad externa: SE1, SE2.

- SE1: Falsificación de voucher.
- SE2: Acceso a la información que viaja por la red.

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios SE1, SE2:

- Canales de transferencia segura: SFTP.
- Estandarizar la ráfaga bancaria.
- Chequeo de la integridad de los datos.
- Definir datos esenciales de un pago.

Realización de los escenarios:

Satisfactoria, los canales SFTP son mecanismos seguros para la transferencia de información en la red. SE1 tiene relación estrecha con los escenarios Confidencialidad.

Atributo de calidad: **Disponibilidad.**

Escenarios de Ocurrencia de fallo: F1...F3.

- F1: Deja de funcionar un servidor.
- F2: No hay conexión con los servidores de aplicaciones.
- F3: Fallo en la conexión con la base de datos.

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios F1...F3:

- Políticas de recuperación ante caídas de servidores: servidores de respaldo, servidores de réplicas.
- Tratamiento de excepciones.
- Separaciones funcionales de los componentes.

Realización de los escenarios:

Estos escenarios no han sido probados.

Escenarios de Latencia de datos: L1, L2.

- L1: Ataques de denegación de servicio.
- L2: Alta concurrencia.

Mecanismos arquitectónicos para mitigar la ocurrencia de los escenarios L1, L2:

- Balanceo de las cargas de peticiones a los servidores.
- Políticas de seguridad web: autenticación en el sistema correctamente, asignación de permisos restringidos a usuarios y roles, deshabilitar el acceso a los servicios.

- Uso de *captcha* (*Completely Automated Public Turing test to tell Computers and Humans Apart*)[6]: para identificar usuarios reales.

Realización de los escenarios:

Estos escenarios no han sido probados.

Arribando a conclusiones del método de evaluación de la arquitectura:

Al ser aplicado este método de validación de la arquitectura, mediante la evaluación de los atributos de calidad definidos, se identificaron puntos sensibles y de compromisos:

La confidencialidad de la información que maneja el sistema se puede ver afectada por un mal manejo de los datos contenidos en la misma, fundamentalmente por causa del factor humano, quien en ocasiones hace uso de manera mal intencionada de dicha información o comete error ajenos a su voluntad. Por ello se utiliza un mecanismo convenido entre las partes implicadas (ver Figura 6) para verificar la integridad de los datos que son intercambiados. La seguridad del mismo recae en el nivel de protección que se le de a los códigos de seguridad asignados a cada banco, este tema es de vital importancia, pues la aplicación se puede ver afectada una vez que estos códigos tengan que viajar por la red. Para evitar esta situación se propone el uso del protocolo SFTP.

[6] Texto distorsionado para validar la presencia humano en un momento en el que un sistema informático precisa evitar que otra computadora intente un ataque.

El sistema debe comunicarse con varios bancos recaudadores, cada uno de los cuales funciona de manera independiente a los demás, de ahí que el intercambio de los datos no es de manera estándar, por tal razón se propone la creación de una ráfaga bancaria única que hace que la información de los pagos se pueda intercambiar de la misma forma con todos estos bancos. Estas ráfagas se intercambiarán mediante el *voucher* que portarán los clientes y los ficheros XML que emiten los bancos con la información de los pagos.

La comunicación con el Sistema de Oficinas del SAIME es más flexible que la convenida con los bancos (se puede conocer su negocio y la de los bancos no), además requiere de comunicación en línea para la consulta de los estados de los pagos de sus clientes.

La tecnología o *hardware* que se utiliza para la realización y el mantenimiento de los sistemas informáticos está sujeta a fallos técnicos, en la mayoría de los casos no es posible controlarlos o predecirlos. Esta situación trae efectos negativos en la disponibilidad de los sistemas y la información contenida, de igual forma puede afectar la integridad de los datos, producto a pérdidas en los dispositivos de almacenamiento.

3.4. Conclusiones

Realizar la evaluación utilizando el método ATAM permitió reconocer cuáles son los atributos de calidad que mayor incidencia tienen sobre la arquitectura y cómo se manifiestan entre ellos. Partiendo de este punto se llega a un consenso entre las partes implicadas sobre la mejor vía para mitigar su impacto.

De manera general, la evaluación frente a los atributos es positiva, aunque quedaron pendientes las realizaciones de los escenarios de disponibilidad y, por tanto, la probabilidad de cambios en función de solucionar posibles inconvenientes surgidos. Estos cambios no se consideran significativos, pues su propagación en el resto del sistema tiene impacto mínimo.

El capítulo evidencia cómo los mecanismos de comunicación con los sistemas externos al SIB permiten un grado de interoperabilidad que es fundamental para la adaptabilidad ante la variedad de orígenes de estos sistemas externos.

CONCLUSIONES GENERALES

La Arquitectura de Software para el Sistema de Integración Bancaria de la Dirección de Gestión Administrativa del SAIME, propuesta a través del presente trabajo de investigación, permite el desarrollo exitoso del mismo, pues facilita la comunicación entre los sistemas que deben integrarse y brinda la seguridad necesaria para realizar los trámites de pago que se deseen.

El ambiente de desarrollo definido a través de las tecnologías, herramientas y metodología seleccionadas constituye una guía para la implementación del SIB.

La evaluación de la arquitectura permite valorar las decisiones arquitectónicas teniendo en cuenta los escenarios de alta prioridad que influyen en ella.

RECOMENDACIONES

- Añadir un módulo al subsistema “Solicitud de Pago” que permita, a los clientes del SAIME, realizar transferencias monetarias asociadas a los trámites bancarios a través de la Web, ya que actualmente los sistemas de pagos en línea son los más populares por las facilidades que brindan.
- Probar los escenarios de disponibilidad de manera que se pueda determinar si la propuesta de arquitectura mitiga los escenarios de alta prioridad que puedan incidir en ella.

BIBLIOGRAFÍA

1. Negocios por Internet. [En línea] SwiftThemes.Com. [Citado el: 12 de Enero de 2011.] http://comohacernegociosporinternet.com/.

2. WebTaller. [En línea] Factoría de Internet, 2003. http://www.webtaller.com/manual-java/caracteristicas-java.php.

3. Group, The PHP. PHP. [En línea] [Citado el: 13 de Enero de 2011.] http://www.php.net/.

4. EVA. *Conferencia #2 El lenguaje XHTML. Principales etiquetas.* [En línea] [Citado el: 14 de Enero de 2011.] http://eva.uci.cu.

5. Programación en Castellano. *Introducción a UML.* [En línea] [Citado el: 15 de Enero de 2011.] http://www.programacion.com/tutorial/uml/1/.

6. R. Baeza-Yates, B. Ribeiro-Neto. Modern Information Retrieval. Harlow : Addison Wesley, 1999.

7. Fabien Potencier, François Zaninotto. *Symfony la guía definitiva .* 2008.

8. Posicionamiento Web. [En línea] Developers4Web.com. http://posicionamientobuscadores.developers4web.com/css-ventajas-al-posicionamiento.

9. CRAIG WALLS, RYAN BREIDENBACH. *Spring in Action.* s.l. : Manning Publications Co.

10. Javi's Java. [En línea] WordPress. http://www.javisjava.com/blog/quartz.

11. Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, y Steve Ebersole. Hibernate. *Tutorial basico de Hibernate.* [En línea] http://www.davidmarco.es/tutoriales/hibernate-reference/.

12. Visual Paradigm. [En línea] http://www.visual-paradigm.com/.

13. Apache. [En línea] Apache Foundation. http://www.apache.org.

14. DosIdeas. *Web Service.* [En línea] http://www.dosideas.com/wiki/Web_Service.

15. Rick Kazman,Mark Klein,Paul Clements. *ATAM: Method for Architecture Evaluation.* 2000.

16. Colectivo de autores. *Procedimiento para Evaluar Arquitecturas de Software.*

17. Carlos Reynoso, Nicolás Kicillof. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2000.

18. David Garlan y Mary Shaw. *An introduction to software architecture.* 1994.

19. D. Garlan and D. Perry. *Introduction to the special issue of software architecture.*

20. Hernández, Ing. Johann Rodríguez. *Especificaciones técnicas sobre la solución de software de integración entre los Bancos recaudadores y el SAIME.* La Habana : s.n., 2010.

21. Jayasinghe, Deepal. *Quickstart Apache Axis2.* Mumbai : Packt Publishing Ltd, 2008. 978-1-847192-86-8.

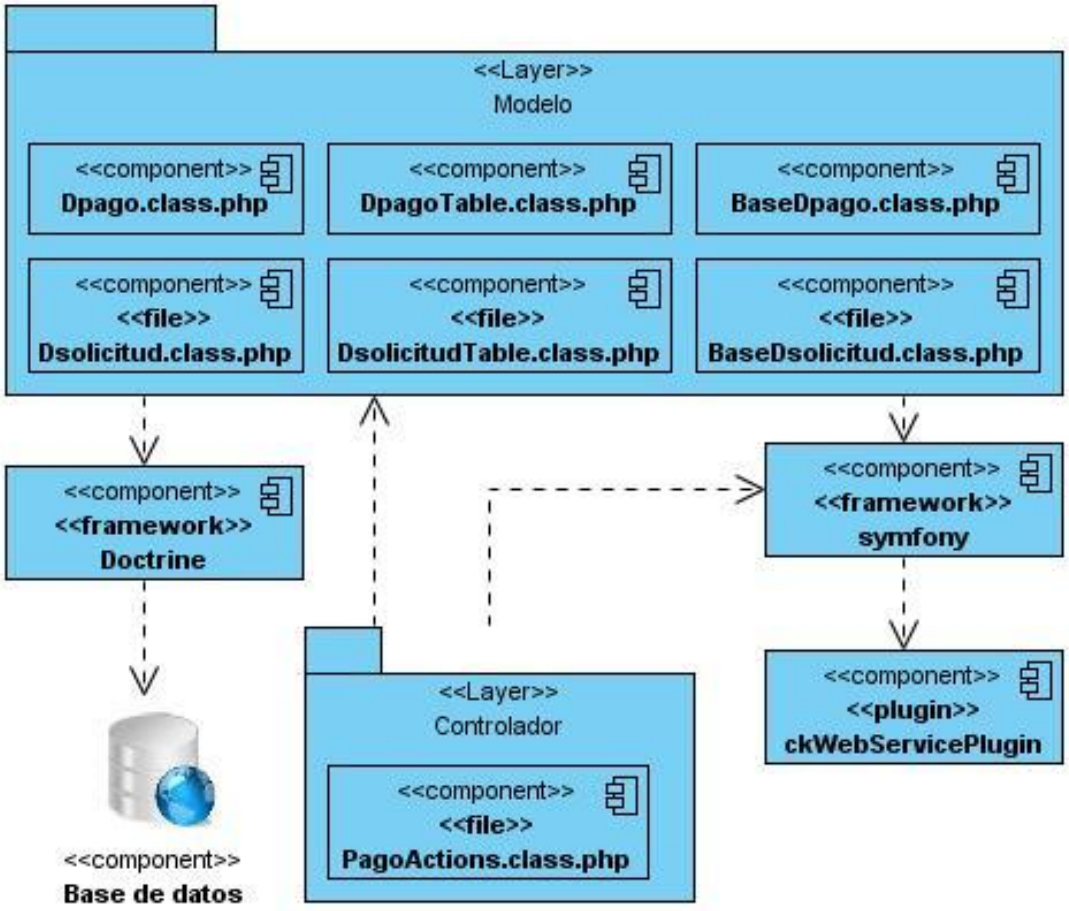
22. PostgreSQL-es. [En línea] 2011. http://www.postgresql.org.es/sobre_postgresql.

23. D. Garlan, D. Perry. *Software Architecture: Practice, Potentials and Pitfalls.* 1994.

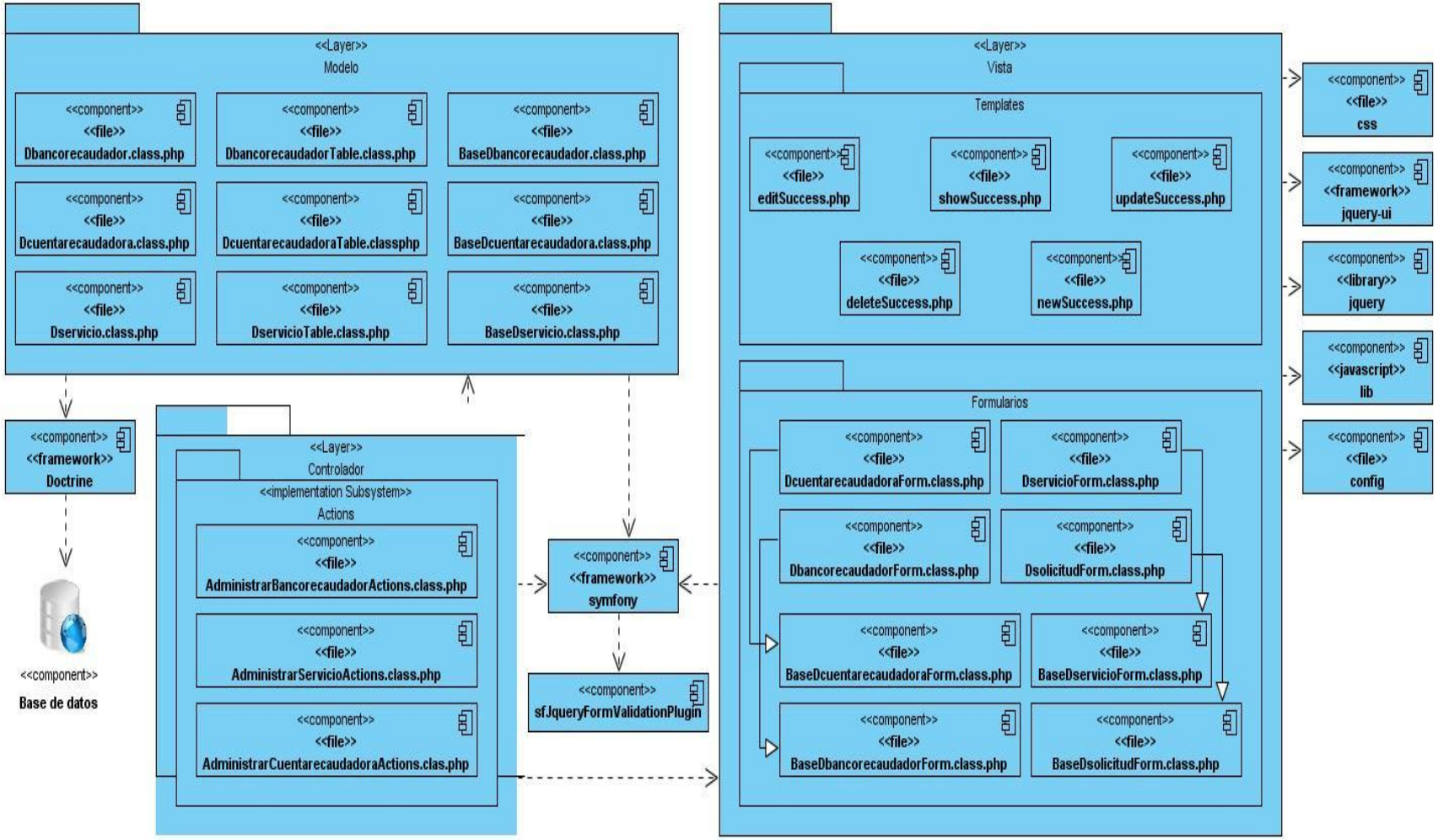
24. García Rubio, Félix Oscar. C. B. S. "Metodologías de Desarrollo de Software". [En línea] http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/tema3_1xh.pdf.

ANEXOS

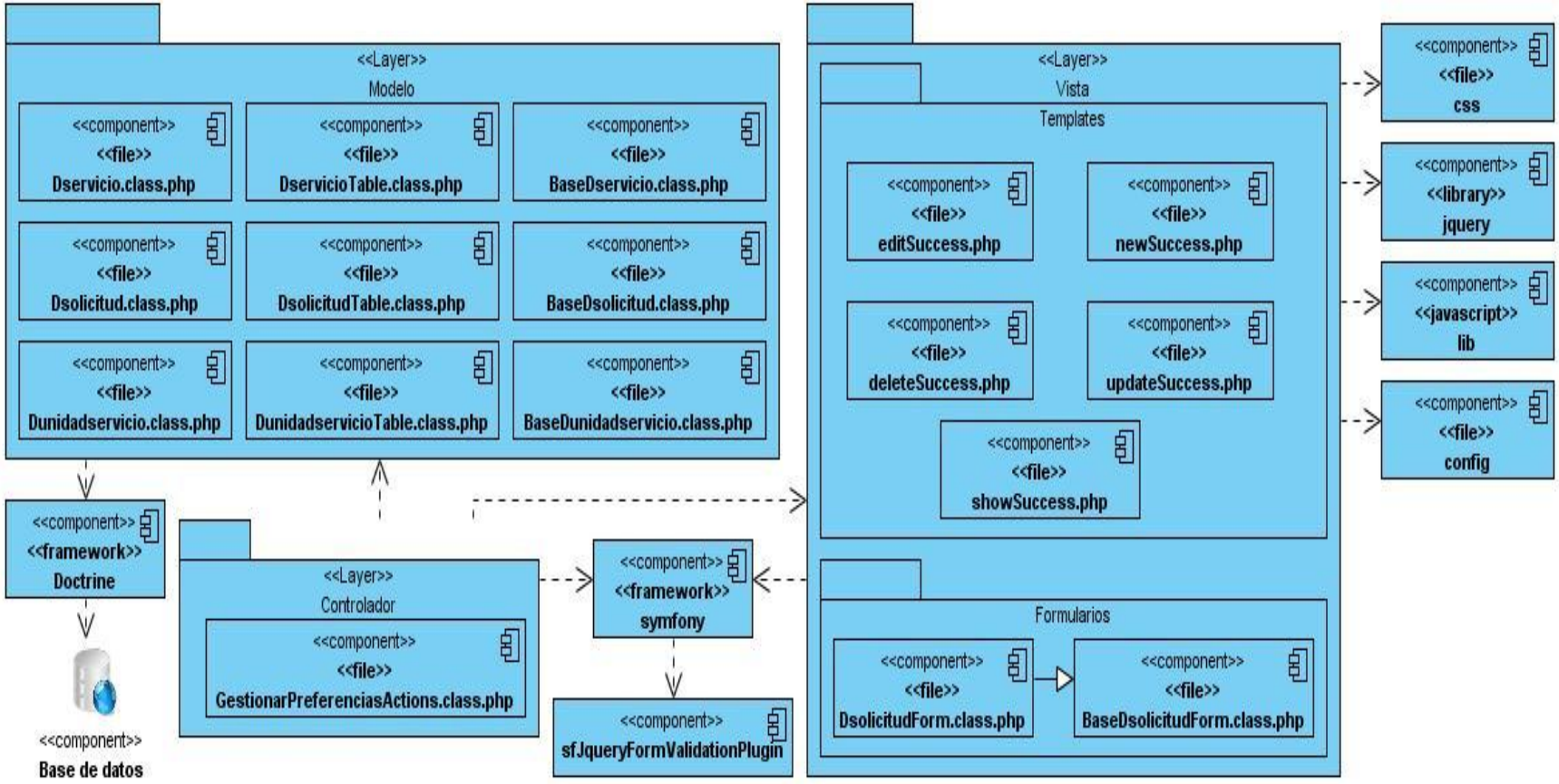
Anexo 1 Vista de implementación del módulo Pago.



Anexo 2 Vista de implementación del módulo Administración.



Anexo 3 Vista de implementación del módulo Solicitudes.



Anexo 4

Documento: Especificaciones Técnicas para la Integración con el Sistema de Oficinas del SAIME.

Anexo 5

Documento: Especificaciones técnicas sobre la solución de Software de integración entre los Bancos recaudadores y el SAIME.