Extensión de Symfony para el desarrollo de aplicaciones en el Sistema de Información para la Salud.

Tesis presentada en opción al título de
Máster en Ciencias en Informática Aplicada

Autor: Ing. Renier Ricardo Figueredo
Tutor: Dr C. José Ortiz Rojas

Ciudad de la Habana diciembre de 2010
"Año 52 de La Revolución".

DECLARACIÓN JURADA DE AUTORÍA Y AGRADECIMIENTOS

Yo Renier Ricardo Figueredo, con carné de identidad 80060720648, declaro que soy el autor principal del resultado que expongo en el presente trabajo de diploma titulado “sISISaludPlugin, extensión de Symfony para el desarrollo de aplicaciones en el Sistema de Información para la Salud”, para optar por el título de Máster en Informática Aplicada. Este trabajo fue desarrollado durante los años 2009-2010.

Deseo agradecer al Dr C José Ortiz Rojas, quien fungió como tutor en mi formación como máster. Agradecer además, a todo el colectivo de la Dirección de Investigaciones, en especial al Dr C Jorge Gulín González por su ayuda y por el tiempo dedicado en la elaboración de esta tesis. A mi compañera Yunaysy por apoyarme siempre. A todos ellos, así como a otros colegas y amigos que no menciono por razones de espacio, les doy las más sinceras gracias.

Finalmente declaro que todo lo anteriormente expuesto se ajusta a la verdad y asumo la responsabilidad moral y jurídica que se derive de este juramento profesional.

Y para que así conste, firmo la presente declaración jurada de autoría en Ciudad de La Habana a los __ días del mes de _____ del año ____.

Resumen

Como parte de la informatización de la sociedad cubana se desarrolla el Sistema de Información para la Salud (SISalud), proyecto que permitirá facilitar la gestión de la información relacionada con el sistema de salud cubano.

El SISalud se comporta con un portal de aplicaciones que colaboran entre sí, a través del intercambio de información, tomando como centro del proceso al paciente. En este esfuerzo se encuentran implicadas varias instituciones del país, entre las que se encuentra la Universidad de las Ciencias Informáticas y específicamente la facultad 7.

La mayoría de las aplicaciones que se han implementado utilizan el lenguaje PHP, por lo que la comisión de arquitectura de la facultad 7 con el objetivo de lograr homogeneidad, decidió que se utilizará como *framework* de desarrollo el symfony debido a las potencialidades que posee.

Escoger un *framework* de desarrollo no es suficiente, debido a que ninguno -por muy robusto que sea-, se puede adaptar de manera directa a las particularidades de un proyecto, por lo que la necesidad de cambiar o extender su funcionamiento es generalmente inevitable. En el caso del SISalud existen varios requerimientos que no son cubiertos por el symfony, por lo que fue necesaria la implementación del componente sISISaludPlugin.

El objetivo fundamental de este componente es cubrir los requerimientos horizontales que afectan a todas las aplicaciones del SISalud, de manera que se pueda potenciar la reutilización de código.

Entre los requerimientos fundamentales que se le dio solución se encuentra, la integración entre los componentes y el control de seguridad a través de la integración con el SAAA (Sistema de Autenticación Auditoría y Acceso). Otro de los aportes del trabajo es la utilización de la Programación Orientada a Aspecto para la realización de la auditoría de las operaciones realizadas por los usuarios.

Índice

Índice de figuras

Introducción

El sistema de salud cubano es uno de los sectores más beneficiados por la revolución, siendo junto a la educación, una las principales conquistas. Ha sido una carta de presentación en distintos países del mundo y es el sector en el que más cooperación se establece con otros pueblos hermanos. Actualmente la solidaridad cubana en la esfera de la salud es reconocida a nivel mundial, especialmente en América Latina, donde ha tenido gran incidencia. El reconocimiento mundial es tan grande que tuvo que ser reconocido por el presidente de los Estados Unidos de América en la “5ta Cumbre de las Américas” en Trinidad y Tobago en el año 2009.

El Sistema Nacional de Salud (SNS) se estructura en tres niveles que se corresponden con la estructura político-administrativa del país. El nacional está representado por el Ministerio de Salud Pública (MINSAP), que es el órgano rector del sistema de salud. Además, a este nivel se encuentran otras entidades de alcance nacional como son centros de investigación, centros universitarios u hospitales de asistencia médica altamente especializados. Los otros dos corresponden a las Direcciones Provinciales y Municipales de Salud, donde cada cual agrupa las instituciones de salud en sus respectivos niveles (1).

Como parte de la batalla de ideas se comienzan a trazar estrategias para aumentar la calidad de los servicios médicos. En este contexto el uso de las Tecnologías de la Informática y las Comunicaciones (TIC) tiene un alto impacto, al proporcionar herramientas para la gestión de la información que se genera. Con este objetivo a partir de 1996 se dan los primeros pasos para el uso y desarrollo de las TIC en el país y en 1997 la Resolución Económica del V Congreso del Partido Comunista de Cuba refleja orientaciones precisas para trabajar en ese sentido y el Gobierno aprueba, por primera vez, los Lineamientos Generales para la Informatización de la Sociedad, con objetivos generales hasta el 2000 (2).

Actualmente el sector de la salud es uno de los más beneficiados con el proceso de informatización de la sociedad cubana, contando con su propia red informática INFOMED, que abarca a todas las unidades de salud en todo el territorio nacional (3). Con este objetivo se ha comenzado el desarrollo de distintos sistemas que gestionan la información generada en los distintos niveles, permitiendo aumentar la calidad de los servicios que se prestan en el país.

Durante los últimos años, instituciones cubanas han desarrollado sistemas para la informatización de la salud. No obstante, estas soluciones carecían de integración lo que impedía su generalización. Es por esta razón que el MINSAP a partir de 2003 comienza el desarrollo de un sistema que integre la información de las diferentes aplicaciones de salud. Para lograrlo, todos ellas deben formar parte del programa general de informatización del SNS, surgiendo la necesidad de desarrollar una plataforma que

garantice estas metas de una manera sencilla y eficiente, comenzando así el desarrollo del Sistema de Información para la Salud (SISalud) (1).

El SISalud integra las distintas aplicaciones a través de una arquitectura orienta a componentes y orientada a servicios (1) (4) y en el desarrollo de estos sistemas no solo está vinculada la Universidad de las Ciencias Informática (UCI), sino también otras empresas del Ministerio de las Informáticas y las Comunicaciones como son: Desoft, Softel, PcMax, Sys, INFOMED, CEDISAP (5).

SISalud abarca los siguientes sistemas: Administración, Registros Básicos y Codificadores, Atención Médica para los niveles de Atención Primaria, Secundaria y Terciaria, Ayuda a la decisión, entre otros (4).

Cada aplicación que se desarrolle debe ser diseñada teniendo en cuenta, en primer lugar, la integración con los demás componentes ya desarrollados, aspecto que elimina la duplicidad y garantiza oportunidad y precisión de la información. Además, evita que cada aplicación se convierta en una isla de información con utilidad y beneficios limitados (5).

La integración de los sistemas permite un proceso de captura, registro, procesamiento, validación y análisis de la información de forma eficiente, lo que incrementa su consistencia, veracidad y oportunidad. Lo anterior redunda finalmente en el mejoramiento de la actividad administrativa, asistencial, docente y de investigación. Es esta integración la que permite hablar de informatización en el sector de la salud pública cubana y no de proyectos aislados (5).

SISalud se puede definir como un portal de aplicaciones, que conforma su estructura según los derechos de los usuarios que se autentican, dándole solamente acceso a los módulos o aplicaciones a las cuales tiene derecho y mostrándole los avisos personalizados que le corresponden. Proporciona además, una integración estratégica y de acción, que asegura una adecuada introducción de las TIC en todo el sistema de salud (1).

La UCI y específicamente la facultad 7 se encuentran entre las instituciones a nivel nacional que trabajan con el objetivo de informatizar el SNS. Uno de los proyectos pioneros en este proceso es el de Atención Primaria de Salud (APS), el cual ha alcanzado importantes resultados que han permitido guiar el desarrollo de otros proyectos. Este ha sido escogido para definir la línea base de la arquitectura del SISalud y en un primer momento todas las aplicaciones se desarrollaban dentro de este grupo.

Con la incorporación de otras líneas de investigación que salían del alcance definido para APS se crearon otros proyectos como la Red Cubana de Nefrología, Control Sanitario Internacional, Balance Material, entre otros. Esto provocó la necesidad de crear un documento rector de la arquitectura del SISaud, que pautara las normas, contratos y estándares que debían garantizar las aplicaciones que se desarrollaran, garantizando en todo momento la interoperabilidad entre los sistemas.

En este documento se define que el SISalud tendrá una arquitectura orienta a servicios y a componentes, donde cada aplicación que se desarrolle se añadirá como un componente más, garantizando así la integración de toda la información. Pero solo se limita a definir los lineamientos generales, entre los que se destaca la utilización de forma obligatoria de estándares abiertos y de software libre, además de la utilización del Sistema de Auditoría, Autorización y Acceso (SAAA) como el sistema de seguridad, dejando abierta la posibilidad de decidir en los grupos de proyectos la tecnología y metodología a utilizar siempre y cuando cumpliera con las normas establecida.

Específicamente para la facultad 7 el documento anterior no era suficiente, debido a que se necesitaba refinar aún más los requerimientos de arquitectura para dar solución a los siguientes problemas (6).

- Variedad de Entornos de Desarrollo para los proyectos, incluso entre aquellos que tienen las mismas características arquitectónicas y estructurales.
- En algunos casos se identificó el uso de herramientas, componentes y tecnologías privativas lo que conspira con la Estrategia de Migración hacia el Software Libre de la Universidad, Facultad y del MINSAP.
- No existe una estrategia coherente y natural para la integración entre los sistemas, que permita lograr la reutilización de código entre los distintos proyectos, lo que provoca que se repitan soluciones en cada grupo de desarrollo.
- No existe una estandarización en cuanto a los estilos de codificación y nomenclatura para Elementos de Configuración de Software (ECS).

Con este objetivo se creó en la facultad 7 el Grupo de Arquitectura y Tecnologías (año 2007), que tuvo como primera tarea la creación del Documento de Arquitectura de Software donde se definieron los estilos arquitectónicos, tecnologías, estándares de comunicación y codificación, lenguajes de programación, herramientas, licencias de software a utilizar, entre otros elementos que permitió la estandarización de la producción de software dentro la facultad.

A pesar de los esfuerzos realizados aún persisten problemas que no han tenido solución y que impiden lograr la verdadera industria de software que necesita el país. Como resultado de ello, los proyectos aun utilizando la misma tecnología y metodología continúan funcionando en su gran mayoría como “islas” separadas que intentan dar soluciones a problemas comunes, multiplicando los esfuerzos para alcanzar la solución. Cuando existe comunicación entre ellos, se realiza de forma espontánea y por iniciativa propia de los desarrolladores, debido a la no existencia de una estrategia coherente que canalice la comunicación y lo que es más importante, la cooperación entre los distintos grupos de desarrollo. Además, no se cuenta con un marco que permita divulgar, generalizar y probar soluciones que sean útiles para toda la comunidad de una forma eficiente en los distintos proyectos que se ejecutan.

La mayoría de las aplicaciones que se desarrollan para el SISalud son software de gestión que utilizan las mismas tecnologías y deben cumplir con los lineamientos

arquitectónicos que se han definido para este tipo de aplicaciones. Además, existe una serie de requerimientos funcionales y no funcionales que son comunes a todas independientemente del negocio que gestionen; por ejemplo: todas las aplicaciones deben integrase a las ya implementadas para el SISaldud, cumpliendo con las políticas de seguridad; deben implementar interfaces *web service* y la gestión de la seguridad debe hacerse a través del SAAA.

A pesar de ser problemas comunes, cada grupo de proyecto intenta darle solución de forma individual, lo cual provoca las siguientes dificultades:

- Se duplica el esfuerzo al tener varias personas trabajando sobre el mismo tema.
- Aumenta el costo de las aplicaciones al implementar funcionalidades que no dependen directamente del proyecto.
- No se obtienen aplicaciones uniformes debido a que la arquitectura interna de cada una de ellas es diferente.
- Se dificulta la gestión efectiva del conocimiento, debido a que cada proyecto se desarrolla independiente a los demás y el conocimiento generado en cada uno de ellos no es generalizado.
- Es complejo encontrar la solución óptima que pueda ser generalizada.
- Cada elemento desarrollado funciona generalmente en el contexto del proyecto para el cual fue desarrollado y no garantiza su funcionamiento fuera de este.
- La no reutilización de código impide obtener soluciones probadas en distintos ambientes de desarrollo.

Estos problemas persisten por no existir un marco de cooperación entre los proyectos que permita la gestión de conocimiento y la reutilización de código. Con esto se lograría que a los problemas horizontales -que interesan a todos los proyectos- se les diera solución de forma única por la comunidad de desarrollo del SISalud y de esta forma los grupos de desarrollos pudieran concentrarse en dar solución a los problemas particulares de cada proyecto.

Esta cooperación para que sea efectiva, debe ser tanto de tipo tecnológico como humano, por lo que se deben crear las estructuras organizativas y el soporte técnico necesario que permitan alcanzar este propósito. Esto permitirá definir objetivos a corto y a mediano plazo y alcanzar la meta final que es la creación de la industria de software que se necesita.

Por todo lo anterior, en la presente tesis de maestría se identificó el siguiente **problema científico**: ¿cómo garantizar que se gestionen de forma única los requerimientos horizontales de las aplicaciones del SISalud que se desarrollan sobre el *framework* symfony?, siendo el **objeto de estudio** de la presente investigación, el desarrollo de aplicaciones de gestión.

Para dar cumplimiento al problema antes mencionado se plantea como **objetivo general**: desarrollar un *plugin* para el *framework* symfony que contenga los principales elementos reutilizables en cada una de las aplicaciones desarrolladas para el SISalud, de modo tal

que se garantice la implementación de forma única de los requerimientos horizontales de las mismas. Por tanto, el **campo de acción** se centra específicamente en el desarrollo de aplicaciones de gestión para el SISalud.

Teniendo en cuenta lo anterior, se plantea la siguiente **hipótesis científica**: el desarrollo de un *plugin* para el *framework* symfony que contenga los principales elementos reutilizables en cada una de las aplicaciones desarrolladas para el SISalud, permitirá garantizar que se implementen de forma única los requerimientos horizontales de dichas aplicaciones, permitiendo la disminución de esfuerzo y tiempo de desarrollo, además de servir como base para la gestión del conocimiento en la comunidad del SISalud.

Para el desarrollo de la investigación se definieron las siguientes **tareas de la investigación**:

- Valorar el estado actual del desarrollo de aplicaciones de gestión en el Centro de Informática Médica.
- Identificar los problemas comunes que se presentan en todos los grupos de desarrollo de aplicaciones para el SISalud.
- Analizar la arquitectura general del SISalud.
- Analizar los lineamientos de la arquitectura definida para los proyectos de la facultad 7.
- Analizar las tecnologías necesarias para dar cumplimiento al objetivo de la presente investigación.
- Realizar el análisis y diseño de la solución propuesta.
- Implementar el componente slSISaludPlugin.
- Probar el componente SISaludPlugin.
- Validar la solución propuesta por un comité de expertos.

El presente trabajo está estructurado en tres capítulos. En el primero se hace un análisis teórico sobre el estado del arte relacionado con el tema de la investigación. Se brinda una breve explicación sobre la Arquitectura Orientada a Servicio y la Arquitectura Orientada a Componentes que son los estilos arquitectónicos con los que está desarrollado el SISalud. Se da una explicación general sobre la programación Orientada a Aspecto y sobre algunas librerías que actualmente están disponibles para la implementación de este paradigma en PHP. Se explican las principales características del *framework* symfony sobre el cual está desarrollada la propuesta de solución y finalmente se abordan las principales características del SISalud.

En el segundo capítulo se hace la descripción de la propuesta de solución. Se abordan cada uno de los elementos que fueron implementados, así como una explicación de cómo pueden ser utilizados.

En el tercer y último capítulo se explica el proceso de validación por el comité de expertos a través del método DELPHY y los resultados obtenidos en las distintas etapas.

Finalmente se insertan las conclusiones generales, las recomendaciones y los anexos con el fin de esclarecer algunos de los temas abordados en el trabajo.

Capítulo 1: Fundamentación Teórica

En el presente capítulo se brinda una descripción general del estado del arte relacionado con las tecnologías para el desarrollo de aplicaciones web, además del marco teórico que fundamenta el objeto de estudio de la investigación. Se hace una descripción de la arquitectura orientada a servicios y a componentes, debido a que es el patrón utilizado en la implementación del SISalud. Se explican además, los conceptos fundamentales de la Programación Orientada a Aspectos (AOP por sus siglas en inglés) -por ser un punto esencial en este trabajo- y las variantes disponibles para su implementación en PHP. Se exponen las principales características del *framework* symfony, dado que la propuesta de solución se hace sobre esta tecnología; y por último se expone la estructura y funcionamiento del SISalud.

Arquitectura orientada a servicios

En el presente, el principal uso que se le da a internet es a través del acceso interactivo a documentos y aplicaciones. En la mayoría de los casos, estos accesos son realizados por personas que utilizan navegadores, aplicaciones de audio y otros. El poder de la Web puede aumentar significativamente si se extiende la comunicación también entre sistemas (7).

Con el propósito de lograr la interoperabilidad entre las aplicaciones, varias empresas han impulsado distintos tipos de soluciones, que si bien han logrado su objetivo, tienen el inconveniente de que dependen de la tecnología sobre la cual están implementadas, por lo que resulta difícil integrar aplicaciones desarrolladas sobre plataformas diferentes (8).

Para alcanzar una verdadera integración, independientemente de las tecnologías de los sistemas particulares, surge entonces la Arquitectura Orientada a Servicios (SOA) que la OASIS define como:

> *La Arquitectura Orientada a Servicios es un paradigma para organizar y utilizar capacidades distribuidas que pueden estar bajo el control de los diferentes dominios de implementación* (9).

SOA ha surgido con el objetivo de hacer más con menos recursos. Permite hacer la reutilización y la integración mucho más fáciles, reduciendo el tiempo de desarrollo y aumentando la agilidad organizacional. El 80% de las organizaciones están implementando aplicaciones usando SOA con servicios web subyacentes. SOA proporciona mayor flexibilidad para afrontar los cambios tanto en el ambiente de negocios como en la infraestructura tecnológica (10).

En una solución basada en SOA intervienen distintos tipos de aplicaciones que se pueden catalogar en tres tipos dependiendo a su rol en el sistema (11).

- **Clientes**: son los programas que inician la interacción a través del envío de mensajes, provocando la ejecución de alguna funcionalidad en el servicio y puede esperar o no una respuesta.
- **Servicios**: son los programas que responden a las solicitudes realizadas por los clientes, realizando alguna acción en dependencia de la información recibida y constituye la base de la arquitectura SOA.
- **Intermediarios**: son programas que sirven de vía a los mensajes y comúnmente hacen funciones de puerta de enlace, monitor de tráfico o gestión de seguridad. Generalmente son implementados utilizando el patrón *proxy*.

En la actualidad es casi imposible la creación de aplicaciones que no tengan algún tipo de interacción con otras y el desarrollo de sistemas distribuidos gana cada vez más fuerzas. En este contexto es donde obtiene protagonismo la arquitectura SOA, pues define los conceptos fundamentales para lograr este tipo de sistemas (12).

Los principios fundamentales para implementar una arquitectura SOA son (11):

- Las fronteras de un servicio son explícitas: esto significa que un servicio debe ser visto como una caja negra donde el cliente no debe asumir nada sobre su funcionamiento, lo que garantiza su flexibilidad y reusabilidad.
- Cada servicio es autónomo: los servicios deben ser tolerante a fallos y se debe poder adicionar, eliminar o modificar funcionalidades sin que sea necesario detener el sistema completo. Esto parte del principio de que un sistema distribuido siempre está en constante cambio, desarrollo y nunca se puede decir que está completo.
- Los sistemas comparten esquemas y contratos, no clases y tipos: en el intercambio de mensajes entre el cliente y el servicio lo único que intercambian son esquemas (datos) y contratos (funcionalidad). Esto se debe a que en una arquitectura SOA no se conoce el ambiente de ejecución de los agentes.
- La compatibilidad es basada en políticas: los servicios no interactúan de forma arbitraria entre ellos, sino que definen mecanismos para determinar cómo será la interacción. Cada servicio debe describir sus funcionalidades y requerimientos específicos.

Entre las ventajas más significativas de una arquitectura SOA se destaca el bajo acoplamiento entre el cliente y el servicio, debido a que realizar cambios internos en un servicio no implica rehacer o reiniciar el sistema. Los desarrolladores se abstraen de la ubicación del servicio, accediendo a este de igual forma que si estuviera en la misma PC o en una ubicación remota. Los servicios son implementados usando protocolos de aplicación por lo que son independientes a otros protocolos de bajo nivel, y por último, los servicios son independientes de la plataforma en que son implementados. Un cliente debe ser capaz de interactuar con un servicio sin importar la tecnología con la cual fue desarrollado, esta es quizás la ventaja más importante y fue la causa fundamental de su creación (8).

Arquitectura orientada a componentes

El desarrollo de aplicaciones orientadas a componentes (*Component Oriented Programming*, COP por sus siglas en inglés) permite el desarrollo de grandes sistemas al descomponerlos en partes más pequeñas. Un componente entonces estará encargado de solucionar problemas más específicos y la unión de todos dará como resultado el sistema final (13).

El objetivo es identificar y traducir los requerimientos del negocio a una serie de componentes que puedan ser aislados como módulos independientes, definiendo sus interfaces y ensamblándolos posteriormente para obtener una arquitectura software modular y flexible, y a la vez coherente y completa (14).

Un componente además de encapsular su complejidad de implementación, como lo hacen los objetos, expone una interfaz que hace que pueda ser conectado con otras piezas de software sin depender una de la otra, es decir, puede evolucionar de forma aislada sin comprometer la funcionalidad de quienes lo utilizan (15).

Un componente es una unidad básica de implementación donde los cambios internos que se le realicen no afectan la comunicación, siempre y cuando mantenga la interfaz pública que implementa (16).

Este tipo de arquitectura permite la incorporación de nuevos requerimientos del negocio que usualmente no implican cambios en la misma. Las nuevas funcionalidades se podrán implementar en un nuevo componente que sería adicionado al sistema (17).

Cada componente debe tener responsabilidades bien definidas, lo cual permitirá reutilizarlo con mayor facilidad en nuevos proyectos. De esta manera aumenta la productividad y la seguridad de los sistemas, al poder utilizar componentes de software ya probados en desarrollos anteriores (17).

Las ventajas que se obtienen al utilizar COP han permitido su adopción como uno de los estándares en la industria. Este tipo de arquitectura es muy útil a la hora de crear sistemas distribuidos y el uso combinado con la arquitectura SOA permite el modelamiento de grandes procesos del negocio. Siguiendo este enfoque, cada servicio de una arquitectura SOA se comportaría como un componente (12) (18).

Framework symfony

Un *framework* en el desarrollo de software es una estructura de soporte definida, mediante la cual un proyecto puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas o lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto (19).

Los *framework* tienen como principal objetivo maximizar la reutilización de código mediante la implementación de los patrones utilizados para implementar las tareas

comunes. Además, permiten estructurar el código fuente forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un *framework* facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas (19).

El symfony es un *framework* patrocinado por Sensio una agencia Web francesa que liberó su código bajo las políticas de código abierto (20). Es uno de los *framework* más populares en la actualidad (21) y está pensado para optimizar el desarrollo de aplicaciones web. Implementa el patrón modelo-vista-controlador, patrón que se ha convertido en un estándar para el desarrollo de aplicaciones de este tipo (22). Symfony cuenta con una gran cantidad de herramientas y clases que disminuyen el tiempo de desarrollo de aplicaciones complejas. La utilización de este *framework* permite que los desarrolladores se concentren en implementar la lógica específica de cada aplicación, lo cual posibilita aumentar la productividad, está implementado sobre PHP 5 y tiene soporte para los gestores de bases de datos más utilizados como son: MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Además, funciona sobre los sistemas operativos Windows y Linux (19).

Algunas de las características de symfony son las siguientes (19).

- Es fácil de instalar y configurar en la mayoría de las plataformas (con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares)
- Es independiente del sistema gestor de bases de datos.
- Es suficientemente flexible para adaptarse a la mayoría de los casos.
- Implementa la mayoría de las mejores prácticas y patrones de diseño para la Web.
- Está preparado para aplicaciones empresariales y es adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Posee código fácil de leer que incluye comentarios de *phpDocumentor* permitiendo un mantenimiento sencillo.
- Es fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.

Otras de las ventajas de symfony es el uso de un ORM (*object-relational mapping*) para la capa de acceso a datos, lo que facilita el trabajo con las bases de datos relacionales. El uso del ORM permite que los desarrolladores diseñen toda la aplicación con el paradigma de Programación Orientada a Aspecto, dejándole al *framework* la responsabilidad de traducir los datos de la lógica de objetos a la lógica relacional (23).

Symfony a partir de la versión 1.2 permite el trabajo con los dos ORM más populares para PHP: Propel y Doctrine, este último fue el escogido para el desarrollo de las aplicaciones del SISalud debido a las ventajas que tiene sobre Propel (24) (25). Desde el comienzo de su desarrollo, Doctrine fue concebido para integrarlo con el symfony. Actualmente el desarrollador de este ORM es miembro de la empresa patrocinadora de symfony.

Symfony a pesar de ser un *framework* muy completo que se adapta a casi cualquier arquitectura y requerimientos del negocio, en ocasiones es necesario extender su funcionamiento. Es por esta razón que el *framework* es altamente configurable permitiendo la modificación de casi todas las partes del sistema. Además, es posible remplazar las clases de su núcleo por clases propias desarrolladas por los usuarios.

Una de las formas de extender el *framework* es a través del desarrollo de *plugins* que permiten encapsular elementos genéricos para varias aplicaciones. Estos *plugins* pueden ser empaquetados como paquetes PEAR (26) facilitando su distribución entre los desarrolladores. Dentro de un *plugin* se pueden implementar clases, módulos, ficheros de configuración, etc. Básicamente se puede ver como un pequeño proyecto dentro de otro. Es la forma más eficiente de distribuir código reutilizable entre las aplicaciones.

Sistema de Información para la Salud

El SISalud es una plataforma que puede ser clasificada como un portal, que agrupa aplicaciones que informatizan distintos procesos del SNS. Comenzó su desarrollo como parte del proceso de informatización de la sociedad cubana y tiene como centro de información al paciente (4).

Su arquitectura es orientada a componentes y a servicios y las aplicaciones son desarrolladas por instituciones a nivel nacional que cooperan con el objetivo de informatizar el SNS (27).

Con el fin de pautar las normas, definiciones y arquitectura que deben cumplir las aplicaciones que se desarrollen, se creó el Grupo de Arquitectura MINSAP – MIC. Su misión es gestionar la integración entre las aplicaciones y componentes distribuidos que se desarrollan por diferentes entidades y grupos de proyectos del MINSAP y el MIC (28).

Este grupo fue el encargado de dictar las siguientes políticas para la informatización del sector de la salud (28).

- El proceso de informatización responde a las políticas y principios socialistas.
- El desarrollo de producciones, inversiones y donaciones de sistemas informáticos deben responder a estrategias, planes de desarrollo, políticas de estandarización y proyectos que serán aprobados centralmente.
- Todos los productos y servicios se integrarán a la ciberinfraestructura del sector y se realizarán en lo fundamental sobre sistemas abiertos, arquitectura orientada a los servicios y basadas en componentes, utilizando software libre y de calidad.
- Deben constituirse en componentes modulares y estables, que compartan normas y cooperen entre sí.
- La informatización debe alinearse con las tecnologías de punta y los estándares de calidad desarrollados en el mundo y adecuarse a nuestras condiciones particulares.
- La seguridad informática y de contingencia son requisitos imprescindibles y responsabilidad ineludible de los productores, prestadores y usuarios.

- Todas las inversiones y proyectos que se desarrollen en el SNS deben considerar el elemento informático desde su concepción inicial.

Además de las políticas anteriores, el grupo de arquitectura definió los principales requerimientos que deben cumplir cada una de las aplicaciones, con el objetivo de sustentar la arquitectura general del SISalud. Estos requerimientos están validados por más de 15 sistemas que forman parte del Registro Informatizado de Salud (RIS) que actualmente está desplegado en el *cluster* de servidores en el nodo central de INFOMED (28).

Uno de los requerimientos que más influye en los componentes es el de seguridad. Cada uno debe implementar una arquitectura que le permita integrarse con los demás componentes siguiendo ciertas normas en la comunicación. Para esto los sistemas deberán contar con un mecanismo de seguridad basado en el modelo Autenticación, Autorización y Auditoría (AAA), lo que provee un conjunto de herramientas y protocolos que garantizan el tratamiento coherente de la seguridad en las aplicaciones.

La autenticación será la primera acción del usuario en el sistema y consistirá en suministrar un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica (28). Esto permitirá identificar desde el primer momento a los usuarios y poder personalizar el sistema según sus privilegios. La autenticación es la base para poder cumplir con los otros dos elementos del modelo AAA.

Después que un usuario es autenticado, el SISalud le muestra las aplicaciones a las cuales puede acceder según sus derechos. Los sistemas serán responsables de denegar o autorizar las acciones que intente hacer según el rol. No basta que un usuario esté autenticado sino que también se debe garantizar que este solo realice las acciones a las cuales tiene derecho según su función en los sistemas.

Uno de los retos fundamentales de los sistemas informáticos es garantizar vías para comprobar la responsabilidad de los usuarios sobre las acciones que realizan. Este es el objetivo de la Auditoría en el modelo AAA. Los sistemas del SISalud deben registrar trazas con la hora y la descripción de todas las operaciones realizadas, lo que además permitirá mostrar estadísticas sobre el funcionamiento del sistema.

Con el objetivo de centralizar el proceso de seguridad en el SISalud y estandarizar la implementación del modelo AAA en todas las aplicaciones, se desarrolló el SAAA, sistema que se comporta como un componente más que implementa una interfaz *web service*. Este sistema permite gestionar de forma centralizada los usuarios, sus privilegios y las trazas de las operaciones realizadas. Con la utilización de este sistema los demás componentes solo les queda implementar una arquitectura que le permita la integración al SAAA.

Otros de los requerimientos de seguridad que debe ser garantizado es el modelo de "Autenticación de firma única" conocida por el término en inglés "*Single Sign On*". Un usuario después de autenticado en la página inicial del SISalud no necesita volver a

identificarse en las demás aplicaciones, debido a que los sistemas deben ser capaces de intercambiar entre ellos las credenciales de los usuarios.

El SNS se organiza en tres niveles de atención: primaria, secundaria y especializada. La atención primaria es el primer punto de encuentro de los pacientes sanos o enfermos con el SNS. Este servicio se brinda a nivel de área de salud, donde la estructura fundamental de atención es el médico de familia, con su equipo básico de salud, que incluye el médico, la enfermera y la psicóloga. También en este nivel se incluyen los hospitales rurales. La función principal de la atención primaria es la prevención y promoción de salud en las comunidades (1).

La atención secundaria se brinda en instituciones hospitalarias de carácter municipal o territorial. Los problemas de salud que se atienden son moderados o graves. La atención especializada se brinda a nivel de instituciones u hospitales (1).

Debido a la estructura del SNS los componentes que forman parte del SISalud se clasifican en cada uno de estos niveles, con el objetivo de mantener una alta cohesión en los procesos que gestionan. Además, existe otro grupo de componentes que su función es tener centralizada la información útil para la planificación y la toma de decisiones. La siguiente figura muestra la estructura del SISalud (1).

Figura: 1. Estructura del Sistema de Información para la Salud**.**

Programación Orientada a Aspecto

A mediados de los 90 surgió la Programación Orientada a Aspectos (AOP por sus siglas en inglés), un paradigma de programación que ha logrado un nivel de desarrollo interesante y que ha generado expectativas internacionalmente. Este paradigma ofrece una nueva herramienta de organización de código: el aspecto (29).

Los aspectos proporcionan constructores para capturar los elementos que se diseminan por todo el sistema y se define como:

Un aspecto es una unidad que se define en términos de información parcial de otras unidades (30).

Anteriormente la Programación Orientada a Objetos (POO) había permitido un avance importante en el desarrollo de software, permitiendo construir sistemas complejos a través de la descomposición en partes más pequeñas: los objetos (31). Este paradigma de programación tiene la ventaja de potenciar la reutilización del código sobre los paradigmas antecesores y es más sencilla la incorporación de nuevas estructuras de datos a un programa existente (30). Centra el diseño de las aplicaciones en las estructuras de datos y no en la funcionalidad, lo que permite una mayor sostenibilidad de los programas. No obstante, esta filosofía dejó algunos problemas sin resolver, las funcionalidades de alto nivel quedan dispersas o fraccionadas por toda la aplicación (30).

Un ejemplo de funciones de alto nivel pueden ser el código de gestión de la seguridad, trazabilidad de las operaciones o el manejo de errores, que no dependen de la responsabilidad de una clase en particular, sino que deben ser implementadas en todas o varias clases del sistema (32).

Con la programación tradicional no se aíslan las funcionalidades de alto nivel sino que quedan completamente mezcladas con las funcionalidades básica de las clases a través de todo el programa, lo que atenta contra la calidad del software. Se puede decir que en la programación tradicional no se tienen en cuenta las funcionalidades de alto nivel, solo se concentra en la separación de las funcionalidades básicas de un sistema (30).

Para ilustrar el problema se pone el siguiente ejemplo: se desea llevar el control de la seguridad en todos los métodos de las clases que implementan el negocio. Esta es una funcionalidad de alto nivel del sistema que no le corresponde a ninguna clase en particular. Todas las clases y todos los métodos deberían implementarla, lo que provocaría que esta porción de código no pueda ser encapsulada en una única clase (33). Siguiendo la filosofía de la POO todas las funciones deberían, antes de ejecutar su código específico, ejecutar el código de seguridad (34), apareciendo así el primer inconveniente, debido a que con esta forma de trabajo todos los desarrolladores deberían responsabilizarse con la implementación de la seguridad en cada uno de los métodos de las clases, algo bastante engorroso por ser repetitivo y no estar relacionado directamente con el código que se está implementando. Además de ello, podría provocar fallos en la seguridad al poder quedar métodos sin protección.

Otro problema está a la hora de dar mantenimiento al software. Si por algún motivo se debe cambiar el mecanismo de seguridad, posiblemente sea necesario modificar todos los métodos en cada una de las clases, algo bastante ineficiente que pudiera dificultar en el mantenimiento.

La solución estaría en buscar la forma de que el código de alto nivel se introduzca de manera automática en el código básico de los métodos, independientemente de la voluntad de los desarrolladores y pueda ser encapsulado en una estructura aparte.

A este problema se le puede dar solución con la AOP. El código común de todas las clases quedaría encapsulado en el aspecto y este se inyectaría directamente dentro de las clases.

Es válido aclarar que la AOP no pretende romper con la POO sino que la completa y la extiende (35).

Algunos de los conceptos fundamentales que se definen en la AOP son los siguientes[1]:

Aspect (aspecto): es una funcionalidad horizontal o transversal que se debe ejecutar en todo el programa y que puede ser separada de forma modular a través de la AOP. Uno de los ejemplos más comunes de aspecto sería el *loggin* (registro de sucesos) dentro de un sistema, o la seguridad (como el ejemplo anterior).

Join point: es un punto de ejecución dentro del sistema donde el aspecto debe ser conectado. En el ejemplo sería al inicio de la ejecución de cada uno de los métodos.

Advice: es la implementación del aspecto, contiene el código que implementa la nueva funcionalidad.

Pointcut: define los *advice* que deben ser aplicados en cada *join point*.

Librerías disponibles en PHP para AOP

La AOP en PHP aún no tiene un gran desarrollo. Las soluciones que se encuentran todavía no explotan todas las potencialidades de este paradigma y de manera general ningún lenguaje ni tecnología ha logrado alcanzar un alto desarrollo en este sentido (36).

En PHP se encuentran una serie de componentes, que por distintas vías tratan de dar solución al problema, a continuación se expone el funcionamiento de algunos de ellos (37).

phpAspect: fue desarrollado en el año 2006 como parte del proyecto *Google Summer of Code*. Es una librería de clases que extiende la sintaxis de PHP al introducir nuevas palabras claves, por lo que el programador necesita el conocimiento de las nuevas estructuras de programación.

Esta librería enlaza los *aspectos* con la clase en tiempo de compilación, básicamente se comporta como un intérprete de PHP al analizar él directamente los ficheros donde están declarados los *aspectos*, generando una representación XML del código, que luego -usando XSLT- lo transforma nuevamente en PHP. El resultado final es una nueva definición de la clase con la unión de los *aspectos* y la clase original (36).

GAP (*Generic Aspect for PHP*): es liderado por Sebastian Bergmann un activo desarrollador de PHP que tiene entre sus trabajos la creación de *PHPUnit*.

[1] Para conocer el resto de los conceptos se debe revisar la bibliografía especializada.

GAP es otra librería de clases que se ejecuta dinámicamente en tiempo de ejecución a través del uso de la librería *runkit.* GAP intercepta las llamadas a las funciones y le adiciona el código de los *aspectos* que debe ser aplicado en cada caso, generando dinámicamente el nuevo código en la aplicación (36) (38).

Transparent PHP AOP: es una librería de clases que logra una alta abstracción en el uso de los *aspectos*. Esta modifica la declaración original de las clases al introducirle el código que contiene los *aspectos*. No utiliza ningún recurso adicional de PHP y es fácil de configurar. Además, se necesita poco conocimiento extra para su uso, pues con algo de estudio teórico sobre AOP y el análisis de algunos ejemplos, se logra tener una clara idea de su uso.

El funcionamiento básico de *Transparent PHP AOP* es el siguiente: el componente recibe como parámetro el fichero donde está definida la clase a la que se desea aplicar los *aspectos*, uno o varios ficheros XML con la definición de los *aspectos* que se van a aplicar y la dirección donde se va guardar la clase después de ser modificada. Con esto, el componente recompila y genera un nuevo fichero con la definición de la clase y los *aspectos* aplicados. Cuando un desarrollador solicita una clase, el componte carga la clase modificada (con los *aspectos*) y no la definida originalmente. Aprovechando la carga automática de symfony esto se puede hacer de forma transparente al programador, por lo que este no tendrá conciencia directa de su uso, lográndose uno de los objetivos fundamentales de la AOP.

Conclusiones del capítulo

La propuesta de solución deberá tener concebido mecanismos que permitan la integración entre las distintas aplicaciones que integran el SISalud, por lo que se utilizarán servicios web. Además, deberá cumplir con las políticas de seguridad y requerimientos definidos por el grupo de arquitectura MINSAP-MIC.

Se implementará sobre el *framework* symfony, por haber sido el *framework* seleccionado por el grupo de arquitectura de la facultad 7 para el desarrollo de las aplicaciones que usan PHP como lenguaje de programación. La utilización de symfony permitirá aprovechar un conjunto de funcionalidades que posibilitarán aumentar al máximo la reutilización de código y disminución del tiempo de desarrollo.

La implementación de *plugins* es la forma natural que propone symfony para extender el *framework* y compartir código entre distintas aplicaciones. Esta capacidad será utilizada en la propuesta de solución para implementar los componentes comunes a todas las aplicaciones del SISalud.

Otro elemento que debe ser considerado es el modelo de seguridad del SISalud (SAAA). Uno de los aspectos que controla el SAAA es la auditoría de las operaciones ejecutadas por los usuarios. Para lograr esto, se propone la utilización de la AOP, para la cual se usará la librería de clases *Transparent PHP AOP*, debido a las facilidades de configuración que esta brinda.

Capítulo 2: Descripción de la Solución Propuesta

En este capítulo se expone la propuesta de solución. Se realiza una descripción del problema a resolver, se muestra el modelo de dominio junto con la descripción de cada uno de los componentes que lo forman y se muestra el diagrama de casos de usos que guía el proceso de desarrollo.

A continuación se encuentran una serie de epígrafes que describen cada uno de los componentes que conforman el *plugin*[2]. Algunas de las descripciones están acompañadas de ejemplos que ilustran su funcionamiento y uso. Además, se muestran los diagramas de clases del diseño de la solución propuesta.

Características de la solución propuesta

Problema a resolver

La Facultad 7 de la UCI es una de las entidades que desarrolla aplicaciones para el SNS. En esta facultad se agrupan distintos departamentos especializados en determinadas líneas de trabajo y proyectos, según el proceso a informatizar. Esta estructura permite una mejor gestión de los recursos humanos y materiales.

Con el objetivo de organizar la producción en la facultad se creó el grupo de arquitectura encargado de establecer las políticas, tecnologías, procedimientos, herramientas, etc., que serán utilizadas en el desarrollo de los proyectos. Como parte de su función, en 2008 se elaboró el documento de arquitectura que guiaría los desarrollos en la facultad.

Entre los aspectos que fueron definidos fue la utilización del *framework* symfony para aquellas aplicaciones que se implementaran con el lenguaje de programación PHP y como metodología de desarrollo el Proceso Unificado de *Rational* (RUP). Esto permitió que los estudiantes pudieran preparase en el uso de la tecnología desde la docencia, lográndose una mayor productividad.

A pesar de esto, aún persisten algunos problemas. Uno de ellos es la necesidad de crear las condiciones adecuadas para que los requerimientos generales de todas las aplicaciones que se desarrollan para el SISalud, puedan ser resueltos por la comunidad de desarrollo. Ejemplos de estos requerimientos son los relacionados con la integración entre los componentes y la gestión de la seguridad.

Actualmente no existe una adecuada comunicación entre los grupos de proyectos, por lo que cada uno intenta dar solución de manera distinta a los mismos problemas. Generalmente los componentes desarrollados no pueden ser reutilizados entre las aplicaciones debido a que la arquitectura es diferente. Por lo tanto el primer paso para

[2] Para lograr entender completamente el funcionamiento de los *plugins* de symfony, se recomienda realizar previamente un estudio del *framework*.

lograr la reutilización de código es la estandarización de la arquitectura interna de cada uno de los proyectos.

Modelo del dominio.

El modelo de dominio tiene como objetivo construir, catalogar y diseminar un conjunto de componentes de software que tienen aplicación en el software actual y futuro dentro de un dominio de aplicación en particular. El objetivo final es crear mecanismos que les permitan a los ingenieros reutilizar los componentes desarrollados (39). Este tipo de modelo es una alternativa cuando no se tiene un negocio bien definido.

El diagrama del modelo de dominio brinda una representación gráfica de los objetos con sus relaciones que están presentes en el ambiente en que se desarrolla la solución.

A continuación se muestra el modelo de dominio del presente trabajo.

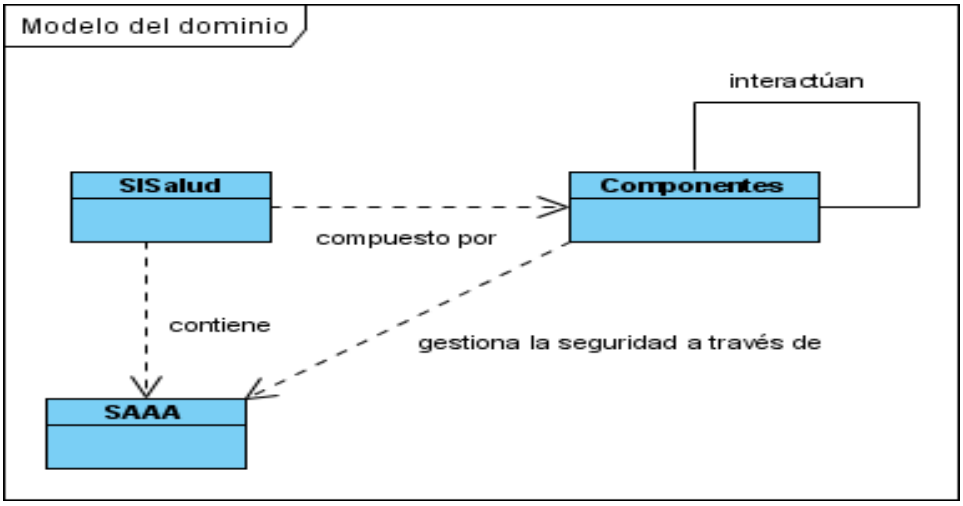


Figura: 2 Modelo del dominio.

Los conceptos que se representan en el diagrama son los siguientes:

SISalud: es el portal de aplicaciones que sirve como plataforma para agrupar todos los componentes.

Componentes: cada una de las aplicaciones que automatizan los distintos procesos y serán desarrolladas por los diferentes grupos de proyectos. Además, estos componentes deben interactuar entre sí para mantener la consistencia de la información que gestionan.

SAAA: componente del SISalud encargado de la gestión de la seguridad en todas las aplicaciones. Está basado en el modelo de Autenticación, Auditoría y Acceso (AAA).

Solución propuesta

El *framework* symfony puede ser configurado y extendido según los requerimientos propios de cada arquitectura. Uno de los mecanismos que propone para lograr estos

objetivos es a través del desarrollo de *plugin* (19). Aprovechando esta característica, se propone la creación del componente slSISaludPlugin que contendrá todos los elementos horizontales útiles para cualquier aplicación que se desarrolle para el SISalud.

El *plugin* puede ser enriquecido por los componentes desarrollados por cada uno de los grupos de proyectos y que no estén acoplados a ningún negocio en particular. Además, permitirá sentar las bases para la adopción de una arquitectura homogénea entre todos los proyectos.

Se define como estándar que todos los elementos que formen parte del *plugin,* como clases, módulos o datos de configuración, deben tener como prefijo "*sf*", lo que permitirá diferenciarlos de los componentes propios del symfony.

Solamente se deben incluir dentro del *plugin* componentes únicamente útiles para el SISalud. Aquellos que puedan ser reutilizados en cualquier aplicación de gestión no deben ser incluidos. El slSISaludPlugin está acoplado al negocio y a la arquitectura del SISalud y está creado para la implementación de los componentes que pertenezcan a este sistema.

Especificación de los requerimientos

La captura de los requisitos tiene como objetivo encontrar las necesidades de los clientes y representarlo de una forma adecuada para que puedan ser entendidos por los usuarios, clientes y desarrolladores. Es importante determinar cuáles son los verdaderos requisitos, pues de ello depende finalmente la satisfacción y aceptación del cliente (40).

En este caso, el objetivo es encontrar cuáles son las necesidades de los grupos de desarrollo a la hora de implementar los diferentes componentes, o sea, determinar aquellos problemas horizontales que afectan en todos los desarrollos. A continuación se listan los requerimientos identificados.

R.1: Gestionar la interfaz de un servicio en una arquitectura SOA del componente.

> **R.1.1:** Permitir la integración al componente directamente por PHP o utilizando servicios web.
>
> **R.1.2:** Validar el *token* de seguridad -enviado en cada petición al servicio- a través del SAAA, independientemente del tipo de conexión.

R.2: Gestionar la conexión con otros componentes.

> **R.2.1:** En el momento de consumir un servicio de otro componente pasarle el *token* de seguridad del usuario autenticado.
>
> **R.2.2:** Configurar las condiciones de integración a los componentes, tipo de conexión y dirección del WSDL, además de los parámetros propios necesarios en cada caso.

R.3: Gestionar los eventos en el SISalud.

R.3.1: Capturar los eventos que se lancen desde el SISalud y convertirlos al modelo de eventos de symfony.

R.3.2: Permitir lanzar eventos hacia el SISalud desde el entorno de ejecución de las aplicaciones.

R.4: Gestionar la integración con el SAAA.

R.4.1: Autenticar a los usuarios dentro de una aplicación desarrollada en symfony a través del SAAA.

R.4.2: Hacer compatible el modelo de asignación de roles de symfony con los roles asignados por el SAAA.

R.4.3: Gestionar ficheros de configuración que permitan normalizar el modelo de seguridad de symfony con el modelo de seguridad del SAAA.

R.4.4: Realizar la auditoría de las operaciones realizadas por los usuarios sobre los sistemas.

R.4.5: Garantizar la filosofía del *single sing on*, en la navegación por las distintas aplicaciones.

R.4.6: Registrar trazas desde las aplicaciones.

Análisis y Diseño

Integración con otros componentes

Uno de los puntos claves en el desarrollo de aplicaciones para el SISalud es la capacidad que deben tener para interactuar con otros componentes ya implementados. Cada una de las aplicaciones informatiza un proceso del SNS y para lograr esto necesita de la búsqueda de información en los demás sistemas. Este intercambio garantiza la consistencia de los datos, además de poder reutilizar lógica del negocio ya implementada por otros componentes (27).

Un ejemplo en el que se evidencia la necesidad de esta interacción es el siguiente: uno de los sistemas que se desarrolla para el SISalud es AlasNefroRed, que inicialmente se encarga de la gestión de la información referente a los pacientes enfermos renales crónicos. En este sistema para registrar un paciente, el usuario debe seleccionar una persona de las registradas en el Registro de Ciudadano (RC) y el sistema debe guardar una referencia de esta persona en su base de datos. Por lo tanto, el AlasNefroRed debe interactuar a través de servicios web con el RC para obtener el listado de los posibles pacientes. Esta forma de trabajo permite conocer toda la información relacionada con un paciente en todos los sistemas pertenecientes al SISalud. Otra interacción que ocurre es

entre los componentes y el SAAA debido a que este último es el sistema para la gestión de la seguridad.

Como ya se ha mencionado, la interacción de los componentes debe seguir las políticas de seguridad establecidas por el grupo de arquitectura MINSAP–MIC. Estas políticas están encaminadas a garantizar que las peticiones a los componentes a través de servicios web, se realicen desde fuentes seguras y por usuarios autorizados. Específicamente plantea, que cada mensaje SOAP que se envíe, debe incluir en la cabecera el certificado de seguridad del usuario autenticado[3].

Otras de las cuestiones importantes a la hora de establecer una conexión con un servicio web es la gestión centralizada de la dirección de los WSDL (41). En PHP para crear un cliente SOAP se utiliza la clase *SoapCliente,* donde uno de los parámetros que recibe en su constructor es la dirección del WSDL del servicio (42). No es correcto poner directamente dentro del código la ubicación del WSDL porque normalmente esta cambia y en ese caso sería difícil el mantenimiento de la aplicación. Lo común es que en tiempo de desarrollo se utilice un servicio de prueba y en tiempo de explotación de las aplicaciones se utilicen los servicios reales.

En una arquitectura SOA no es obligatorio utilizar solo servicios web (43) y en este momento todo el intercambio de información entre los componentes del SISalud se hace a través de esta vía, lo que penaliza el tiempo de ejecución de las aplicaciones. En aquellos sistemas que se ejecuten en el mismo servidor se podría establecer una conexión directamente por PHP lo que permitiría aumentar su rendimiento (12). Además, las aplicaciones deberían ser capaces de conectarse por otras vías como *Socket* o cualquier otra variante que pueda surgir en el futuro.

La solución propuesta da la posibilidad a las aplicaciones de comunicarse utilizando tanto servicios web como directamente por PHP y propone un diseño flexible para la incorporación de nuevas vías en el futuro.

Todas las especificaciones necesarias para lograr la integración deben ser transparentes a los desarrolladores. Su único interés deberá ser ejecutar los métodos que necesite de los demás componentes del SISaldud e implementar los requerimientos de cada aplicación. El objetivo es crear las condiciones que abstraigan al desarrollador de estos requerimientos y se concentre solo en el desarrollo de su solución, con el objetivo de aumentar su productividad.

Debido a que todas las aplicaciones deben integrarse de la misma manera, se han implementado una serie de clases para esto.

En la figura 3 se presenta el diagrama de clases del diseño de la propuesta de solución.

[3] En la sección Integración con el SAAA se brinda información sobre las credenciales de los usuarios, entre cuyos datos se encuentra el certificado.

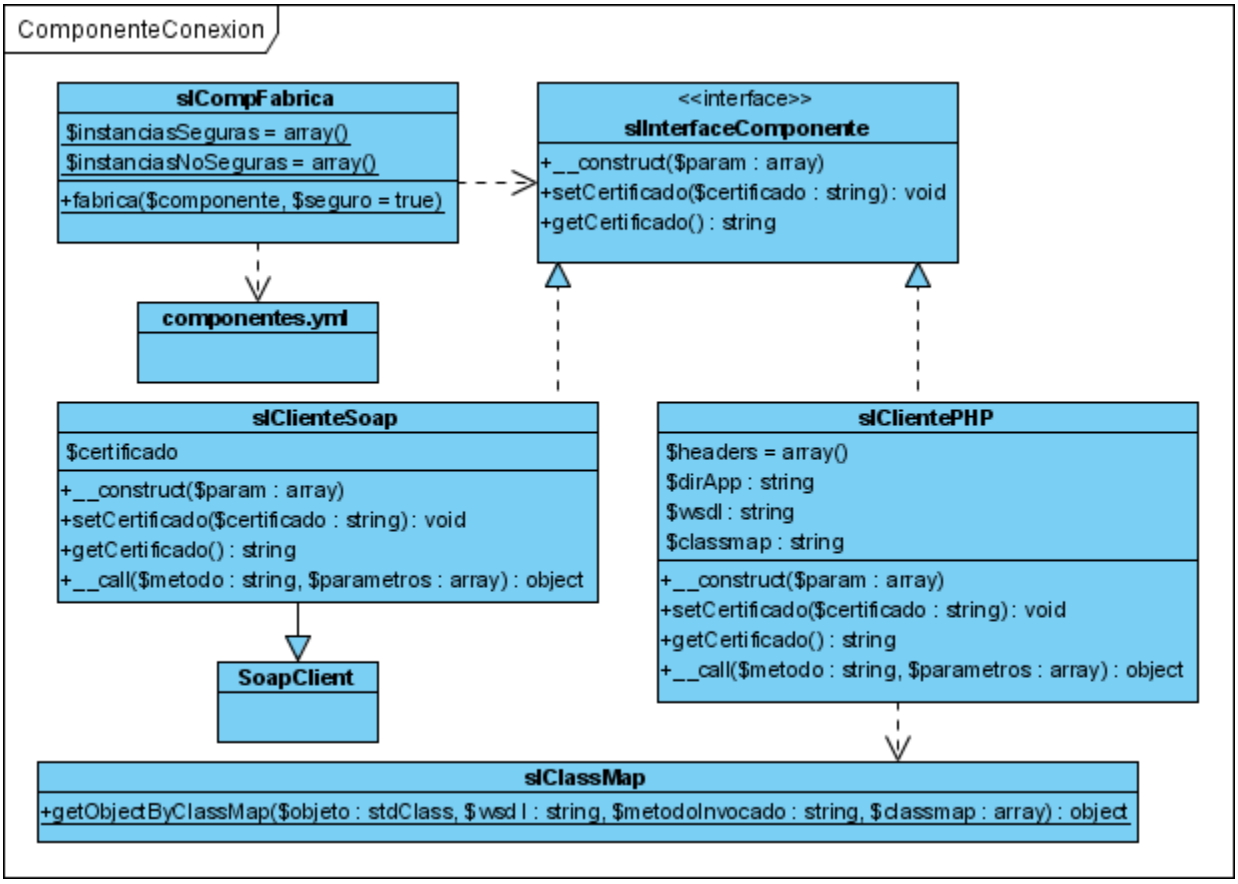


Figura: 3 Diagrama de clases del diseño para la integración entre componentes.

A continuación se brinda una explicación de cada una de estas clases.

La clase *slCompFabrica* es la única clase a la cual tendrán acceso los desarrolladores. Como su nombre lo indica, su objetivo es fabricar la conexión a los distintos componentes. Para esto el único método que posee: *fabrica*, implementa el patrón *abstract factory* (44), que recibe como parámetro el nombre del componente con el cual se quiere establecer la conexión y si esta debe ser segura o no (por defecto es segura). La necesidad de incluir este segundo parámetro está dada porque hay algunos componentes que pueden ser consultados de manera pública sin establecer los mecanismos de seguridad.

Además, posee dos arreglos privados que almacenan las conexiones abiertas en cada momento, una para las conexiones seguras (*instanciasSeguras*) y otra para las no seguras (*instanciasNoSeguras*). Esto no es más que la implementación del patrón *Singleton* (45). Cada vez que se solicita una conexión a una instancia ya creada, se devuelve el objeto almacenado en el arreglo, garantizándose de esta manera una única conexión en toda la aplicación.

El uso de esta clase permite la abstracción de los desarrolladores de los mecanismos de seguridad y del tipo de conexión (46). Lo único que debe conocer es el nombre del componente al cual quiere conectarse. Esta clase hace todo el trabajo y le devuelve una conexión lista para ser usada.

Los requerimientos necesarios para la integración con los componentes se especifican en el fichero de configuración *componentes.yml* ubicado en el directorio *config* del *plugin*, el cual es utilizado luego por la clase *slCompFabrica.*

En el fichero se pone una entrada por cada uno de los componentes junto con las especificaciones necesarias para lograr la conexión. Estas varían según el tipo de conexión que puede ser por servicio web o por PHP.

En el siguiente listado se pone un ejemplo de cómo se configura el fichero *componente.yml* con la explicación de cada uno de los elementos:

Listado 1 Ejemplo del fichero componente.yml.

```
all:
  RC: #Configuración para la conexión con el RC.

    class: slClienteSoap #Esta clase indica que la conexión
                         #es a través de servicio web.

    param:#Parámetros específicos que se le deben pasar a la clase.

      wsdl: http://10.128.52.50:5801/RC/RC.wsdl #Dirección del WSDL.

      opciones: #Como la clase slClienteSoap hereda de SoapClient, es
                #posible pasarle las opciones para la conexión, aquí
                #hay un ejemplo.

        style: <?php echo SOAP_RPC?>
        use:   <?php echo SOAP_ENCODED?>
        classmap:
          Ciudadano: Persona

  NEFRO: #Configuración para la conexión con AlasNefroRed.

    class: slClientePHP #El uso de esta clase indica que la conexión es
                        #directamente por PHP.

    param:
      dirApp: dirección/en/el/disco/duro/server_php.php
                #Dirección del front controller de la otra aplicación.

      wsdl: http://slsisaludplugin:5800/server.wsdl #Direción del WSDL.
      classmap:
         ObjetoRetorno: ObjetoInterno
```

Lo interesante en este fichero es la opción *classmap* (42), que se le pasa como parámetro a ambos tipos de conexiones. Es un arreglo asociativo, donde la llave son los tipos de datos retornados por el servicio y el valor es el tipo de dato al cual debe ser transformado dentro de la aplicación. Esto permite que cuando el desarrollador invoque un servicio, el

valor retornado sea un tipo de dato conocido dentro de su entorno de ejecución. Esta opción no es obligatoria, en caso de no ponerse o no estar definido para un tipo de dato específico, el *plugin* retorna un objeto de tipo *stdClass*.

La interfaz *slInterfaceComponente* (47) tiene definidos los métodos que deben implementar todas las clases desarrolladas para los diferentes tipos de conexión. Define que las clases deben tener un constructor (*__construct*) que recibe un arreglo de parámetros que son los definidos en el campo *param* del fichero *componentes.yml*. Además, define los métodos *setCertificado* y *getCertificado* que permiten establecer y obtener el certificado en las distintas clases.

El uso de esta interfaz permite que la clase *slCompFabrica* no esté acoplada (48) (49) con ninguna clase en particular. El nombre de la clase lo obtiene de la opción *class* del fichero *componentes.yml* y solo interactúa con los métodos definidos por la interfaz. Esto permite que se puedan desarrollar tantos tipos de conexiones como se desee, implementando para cada uno, una clase que implemente la interfaz.

La clase *slClienteSoap* hereda de la clase *SoapClient* (42) de PHP y es la utilizada para las conexiones a servicios web. Lo único nuevo que hace es que en el método *setCertificado* pone el certificado del usuario como una cabecera SOAP a través del método __*setSoapHeaders* de la clase *SoapClient*.

La clase *slClientePHP* es utilizada para la integración directa por PHP, solamente funciona si el servicio está implementado en symfony y utilizando el slSISaludPlugin. En este tipo de conexión lo que se hace es ejecutar el *front controller* de la aplicación que implementa el servicio. Ejecutar la otra aplicación no es tan sencillo como hacer un *include* o un *require* del fichero donde está el código del *front controller*, pues la otra aplicación debe ejecutarse con sus propias clases y configuración.

PHP es un lenguaje de *script* (50), donde el código que se ejecuta va adicionando ficheros en los cuales pueden estar definidas clases y funciones que son cargadas en memoria (42). Esto permite que el código que se ejecute posteriormente, pueda utilizar los recursos que ya están disponibles. PHP no permite que se carguen dos clases con el mismo nombre, aunque estén en ficheros distintos. En estos casos se lanzará una excepción que detendrá el flujo de ejecución de la aplicación. Naturalmente, en dos aplicaciones implementadas con PHP pueden existir clases con el mismo nombre y su lógica puede ser distinta.

Otro problema que impide unir de forma directa dos aplicaciones symfony es que el *framework* implementa lo que se llama carga automática de clases. En symfony, las clases son cargadas en memoria a medida que se van necesitando dentro del código, utilizando la función *spl_autoload_register* (42) de PHP. Para esto guarda en la *cache* del proyecto la ubicación física de cada clase y cada aplicación tiene su propia *cache*.

Para evitar estos problemas se necesita que la aplicación del servicio se ejecute en otro hilo de PHP distinto al de la aplicación cliente. De esta manera se lograría que las dos aplicaciones no compartan el mismo espacio de memoria y cada aplicación se ejecutará con sus propias clases y configuración, sin que existan interferencias entre ellas.

PHP cuenta con la extensión *runkit* (38) (51), que da vías para modificar constantes, clases y funciones en tiempo de ejecución. Además, permite la definición de variables globales y ejecutar dentro de PHP sub-intérpretes a través de lo que se llama *sandboxing*.

La creación de sub-intérpretes se logra a través de la clase *Runkit_Sandbox* que permite crear un nuevo hilo de ejecución con su propia memoria (42). Utilizando el *runkit* y específicamente esta clase, es que se logra integrar la aplicación cliente y el servicio. En la siguiente sección se explica cuál fue la implementación realizada del lado del servidor.

La clase *slClassMap* se utiliza para determinar el tipo de dato al que deben convertirse los datos retornados por los servicios en las conexiones a través de la clase *slClientePHP*. Con esto se logra que esta clase funcione igual que la *slClienteSoap*. De esta manera el desarrollador no notará la diferencia cuando utilice un tipo de conexión u otra. Esta clase puede ser reutilizada en otros modos de comunicación que surjan en el futuro.

Finalmente, para garantizar la completa compatibilidad entre cualquier modo de integración entre aplicaciones, se creó la clase *slConexionExcepcion*, utilizada para reportar errores en la conexión. Cada vía de comunicación puede reportar sus propios tipos de excepciones (por ejemplo: la clase *SoapClient* lanza excepciones de tipo *SoapFault*) y los desarrolladores al no tener conciencia de qué vía se está utilizando en cada momento, todas las excepciones son convertidas a este tipo.

Para ver el modo en que se utilizan estas clases se pondrá el siguiente ejemplo: se desea conectar al RC para consumir un método llamado *getById*[4], el cual recibe como parámetro el *id* del ciudadano y retorna una estructura con los datos de la persona, con un campo para el nombre, la edad y el sexo. Las condiciones de la conexión se pueden ver en el "listado 1". Lo primero sería implementar la clase *Persona* (listado 2).

Listado 2 Implementación de la clase persona.

```
<?php
class Persona {
  var $nombre;
  var $edad;
  var $sexo;
}
```

Luego en cualquier parte del programa el desarrollador podría pedir una conexión para el RC e invocar el método *getById*, vea el ejemplo del listado 3.

Listado 3 Llamada al método *getById*

[4] Este método es hipotético.

```
//en cualquier parte de la aplicación.
$RC = slCompFabrica->fabrica("RC");//se crea una conexión al registro
                                   //de ciudadano de modo seguro.
$persona = $RC->getById(1);//se obtiene el ciudadano con id 1

//se imprimen los datos en pantalla
echo $persona->nombre." ".$persona->edad." ".$persona->sexo;
```

Para que el modo de conexión utilizado en cada momento sea transparente para el desarrollador, todas las vías deben implementar el mismo contrato. Esto significa que los nombres de los métodos, los parámetros y los valores retornados siempre tienen que ser igual. Por lo tanto, sin importar el modo de conexión, el contrato va a ser el WSDL (52) que describe el servicio.

Implementación de la interfaz de servicio de los componentes

El SISalud está desarrollado siguiendo una arquitectura SOA (53), por lo que los componentes deben implementar servicios web que permitan exponer sus servicios. Cuando se desea desarrollar una aplicación de este tipo utilizando PHP, se deben valorar varias alternativas que permitan implementar todos los componentes y tecnologías asociadas de forma óptima teniendo en cuenta el *framework* y las herramientas que se van a utilizar.

Los servicios web también deben garantizar que se cumplan las políticas de seguridad. Cada mensaje SOAP que se reciba debe incluir en la cabecera, el certificado del usuario y los sistemas deben comprobar su validez a través del SAAA.

La solución propuesta en este trabajo permite que los desarrolladores se abstraigan de los mecanismos de seguridad y de la complejidad asociada con el desarrollo de servicios web. Los equipos de trabajo solo deben concentrarse en la implementación de la lógica del negocio de los sistemas.

ckWebServicePlugin

Una de las ventajas del symfony es la gran comunidad de desarrollo que posee. Constantemente se publican en el sitio oficial del *framework* nuevos componentes implementados por la comunidad, generalmente son *plugins* que pueden ser incluidos en las distintas aplicaciones.

Uno de los *plugin* disponibles para symfony es el *ckWebServicePlugin*, el cual permite construir un API de servicios web a partir de una aplicación symfony. Tiene integrado un componente llamado *WSDL-Generator* que genera ficheros WSDL compatibles con otras aplicaciones PHP o clientes .NET o java (54).

Existen herramientas para la edición de WSDL, donde los desarrolladores pueden -generalmente de forma visual-, editar los ficheros WSDL y configurar los distintos

elementos que lo conforman. Este tipo de herramienta tiene el inconveniente de ser difícil de mantener la consistencia entre el servicio y el WSDL que lo describe. Todo esto se debe a que independientemente del nivel de automatización que tenga, el proceso no es completamente automatizado (55).

La generación del WSDL es una de las principales características del *ckWebServicePlugin*, el cual es generado automáticamente del código fuente. Los desarrolladores solo deben marcar las acciones que serán expuestas como servicios y ejecutar un comando para ejecutar todo el proceso (54).

En el siguiente listado se muestra una acción que es expuesta como servicio[5].

Listado 4 Ejemplo de una acción expuesta como un servicio web.

```
<?php
class ejemploActions extends sfActions
{
  /**
   * Método para saludar
   *
   * @ws-enable
   * @ws-header Token: slTokenSaaa
   *
   * @param string $nombre Nombre
   *
   * @return string
   */
  public function executeHola($request)
  {
    $this->result = 'Hola '.$request->getParameter('nombre');
  }
}
```

En este ejemplo se define un servicio web que recibe como parámetro el nombre de una persona y el servicio retorna una cadena de texto. Además, al servicio se le debe pasar una cabecera llamada *Token* (en la siguiente sección se explica su funcionamiento). Todas estas especificaciones se hacen a través de *phpDocumentor* (56) y le permite al generador de WSDL realizar su trabajo.

El trabajo de los desarrolladores se limita a implementar las acciones del servicio y documentarlas con *phpDocumentor*. La creación del servicio web, el WSDL y la gestión de seguridad son implementadas por el *ckWebServicePlugin* y por un grupo de clases pertenecientes al *slSISaludPlugin*. Como se puede observar, implementar un servicio web es prácticamente igual que implementar cualquier otra aplicación, incluso más sencillo debido a que no es necesario implementar las vistas para el usuario.

[5] Para entender el funcionamiento completo de este ejemplo se debe revisar la documentación del *ckWebServicePlugin*.

Gestión de seguridad

Cuando uno de los métodos del servicio web requiera de comprobación de seguridad para ser consumido, debe indicarse que al método se le debe pasar una cabecera con el nombre *Token,* como se ilustra en el listado 4. Esta cabecera será un objeto de tipo *slTokenSaaa* que es una de las clases implementadas dentro de *slSISaludPlugin*.

Cuando en el XML del mensaje SOAP se especifica el paso de una cabecera, el *ckWebServicePlugin* lo convierte al tipo de dato correspondiente dentro de la aplicación y lanza el evento "*webservice.handle_header*" que contiene los datos de la cabecera. Este proceso se hace antes de que se invoque el método del servicio web.

ckWebServicePlugin, para realizar su trabajo, utiliza la extensión SOAP de PHP y específicamente la clase *SoapServer*. Esta clase es la encargada de crear un servidor web sirviendo de fachada entre el XML del mensaje y el lenguaje nativo PHP. Es capaz de procesar dentro del XML el nombre del método y los parámetros que se le pasan al servicio. A pesar de estar supuestamente implementado el tratamiento de las cabeceras SOAP, realmente no funciona y está declarado como uno de los *bug* de esta librería. Esta fue la razón por la que hubo que implementar las clases que se presentan en el siguiente diagrama de clases del diseño.

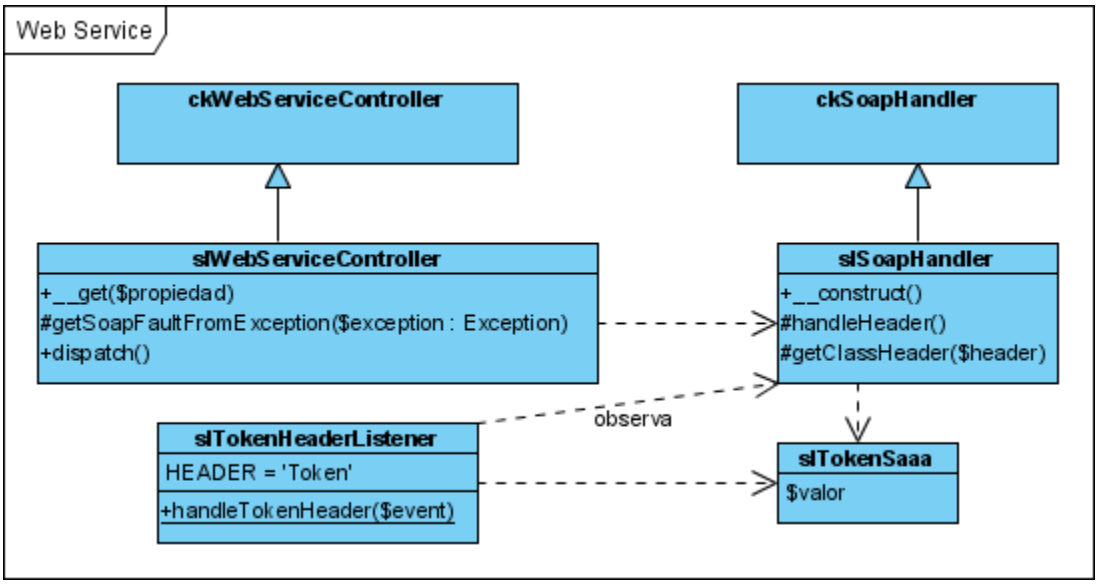


Figura: 4 Implementación de servicio web.

La única función de la clase *slWebServiceController* es exponer mediante una propiedad pública, el XML del mensaje SOAP para que pueda ser utilizado en la clase *slSoapHandler* en el procesamiento de las cabeceras.

Debido a la incapacidad de la extensión SOAP de PHP para el procesamiento de las cabeceras (57), la clase *slSoapHandler* procesa directamente el XML del mensaje. Cada vez que encuentra una cabecera, se la pasa a la clase *ckSoapHandler* para que lance el evento "*webservice.handle_header*".

La clase *slTokenHeaderListener* es la encargada de escuchar el evento para el procesamiento de la cabecera *Token* que debe contener el certificado de seguridad del usuario. Con esta información invoca el método "*autenticarPorCertificado*" de la clase *slUserSaaa*[6]. Si este método retorna falso, significa que el usuario no puede ser autenticado debido a que el certificado es inválido o caduco, en cuyo caso se lanzará una excepción que impedirá la ejecución del servicio web.

Además de todo lo planteado hasta este momento, en la gestión de la seguridad se pueden utilizar todas las variantes que definen symfony y el *plugin slSISaludPlugin,* ya que un servicio web es una aplicación como otra cualquiera.

Estos mecanismos de seguridad garantizan que la ejecución de la lógica del negocio relacionada con un servicio web se ejecute solo si ha pasado por todos los filtros de seguridad, independiente de la voluntad del desarrollador, por lo que este se abstrae de todo este proceso, permitiendo que el servicio se ejecute en un entorno seguro.

Integración mediante PHP

El *slSISaludPlugin* permite la integración no solo a través de servicios web, sino también directamente por PHP, lo que permite ganar en rendimiento en aquellas aplicaciones que se ejecuten en el mismo servidor.

En este tipo de conexión el servicio se ejecuta dentro de un sub-intérprete de PHP mediante *sandboxing* utilizando la librería *runkit*. El cliente debe pasarle el método que debe ser ejecutado, los parámetros y las cabeceras pertenecientes al servicio.

El uso de sub-intérpretes tiene como restricción que lo único que se puede pasar entre los dos hilos de PHP es texto, por lo tanto los parámetros y las cabeceras -que son un conjunto de datos-, deben ser "serializados" mediante la función *serialize* y así convertirlos a cadenas de texto. Con este objetivo el cliente crea tres constantes: *CONEXION_PHP_HEADERS*, *CONEXION_PHP_METODO* y *CONEXION_PHP_PARAMETROS* en el sub-intérprete del servicio, que contiene los valores en formato de texto de las cabeceras, el método y los parámetros respectivamente. Luego de ejecutado el servicio, los valores retornados por los métodos son serializados para devolverlos al cliente. Esta es la razón por la cual -por el momento-, este tipo de conexión solo funciona entre dos aplicaciones implementadas con symfony y que utilicen *slSISaludPlugin*.

La conexión a través de un servicio web o directamente por PHP es igual, excepto por la forma en que se pasan los datos para el servicio. En el primer caso es a través de un XML y en el segundo, es a través de las constantes descritas anteriormente. Por esta razón lo que se hace es extender la funcionalidad del *ckWebServicePlugin* y reutilizar toda la lógica ya implementada en este *plugin*. Lograr que una función exponga sus servicios por cualquiera de las dos vías no implica ningún esfuerzo adicional para el desarrollador, solo es necesario hacer algunas configuraciones.

[6] En la sección Integración con el SAAA se analiza esta clase.

A continuación se presenta el diagrama de clases del diseño para la creación de servicios mediante PHP:

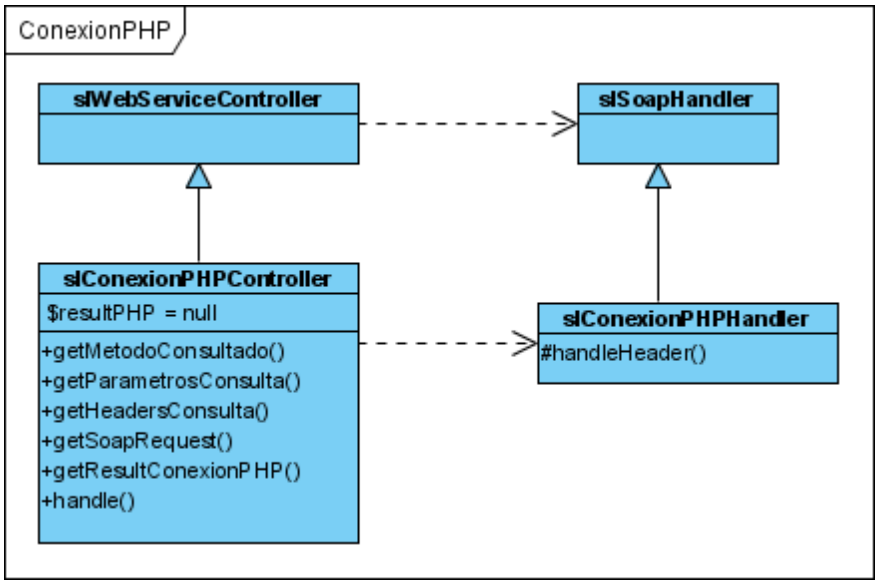


Figura: 5 Diagrama de clases para la creación de servicios mediante PHP.

En este caso solo se han añadido dos clases: *slConexionPHPHandler* y *slConexionPHPController*. La primera se utiliza para procesar las cabeceras pasadas por el cliente y la segunda para obtener los datos necesarios para la ejecución del servicio, que se obtiene desde las constantes mencionadas anteriormente.

La clase *slConexionPHPController* es la encargada de "deserializar" los datos mediante la función *unserialize* y luego serializar el resultado para devolverlo al cliente. Además, captura las excepciones que se lancen desde la ejecución del servicio, que son también serializadas y pasadas al cliente.

Uno de los principios fundamentales en la creación de servicios es que ambas partes solo intercambian esquemas y contratos, no clases ni tipos. Como el cliente y el servicio se ejecutan en dos intérpretes diferentes de PHP, los tipos de datos definidos en un hilo no tienen que estar necesariamente definidos en el otro, y si existen, quizás no estén declarados de la misma forma. Por lo tanto, el paso de datos debe ser de un tipo común en ambas partes. Con este objetivo la clase *slConexionPHPController* convierte todos los valores retornados por el servicio a *stdClass* y todas las excepciones a objetos de tipo *Exception* que son tipos de datos propios de PHP.

Una de las fortalezas de la solución propuesta en este trabajo, está en ofrecer un marco de desarrollo donde lo fundamental es la estrategia a la hora de construir el servicio web y no la complejidad relacionada con la forma de comunicación y construcción. El desarrollador solo se concentra en la implementación de la lógica del negocio. La forma de acceder al servicio no es exclusiva y el nivel de integración e interoperabilidad está garantizada. Construir un servicio utilizando servicio web o directamente por PHP es igual

y solo es necesario hacer algunas configuraciones. El *plugin* ya tiene implementadas tareas que automatizan el proceso de configuración.

sfAspectPlugin para la Programación Orientada a Aspectos

En la sección “Integración de los componentes con el SAAA” se explica la utilización de la AOP para la gestión de la seguridad en los sistemas perteneciente al SISalud. Específicamente se utiliza para realizar la auditoría de las acciones realizadas por los usuarios.

El *sISISaludPlugin* solamente debe incluir aquellos componentes que sean solo útiles para el SISalud. Los que puedan ser reutilizados en cualquier otra aplicación no deben ser incluidos dentro del *plugin*. Este es el caso de la AOP, que puede ser utilizada en cualquier aplicación, por lo que se decidió la creación del componente *sfAspectPlugin* para la implementación de este paradigma, el cual fue publicado en el sitio oficial de symfony para que pueda ser utilizado por toda la comunidad de desarrollo del *framework*.

En el capítulo anterior se expusieron los principales conceptos relacionados con la AOP y las características de “*Transparent PHP AOP*” que es la librería de clases escogida para el desarrollo del *plugin*. Esta librería logra sus objetivos a través de la redefinición de las clases, por lo que se debe buscar la manera de que estas sean modificadas antes de ser usadas. En esta sección se dará a conocer la forma en que fue implementado y diseñado el *sfAsepctplugin*.

Una de las funcionalidades fundamentales del symfony es el mecanismo de carga automática de clases. Significa que el desarrollador no necesita incluir directamente en el código la dirección donde se encuentra definida la clase que desea utilizar, lo único que debe hacer es colocarla en los directorios o subdirectorios específicos que define el *framework* para su ubicación.

Todas las clases definidas en los ficheros que pertenecen a los directorios *lib* del proyecto, de la aplicación, de los *plugin* o de los módulos, se verán beneficiadas de esta funcionalidad, pues esta es la configuración por defecto que se define en el fichero *autoload.yml*, aunque este comportamiento puede ser modificado a conveniencia del desarrollador.

Este proceso se hace a través de la clase *sfAutoload*. El *framework* utiliza la función de PHP *spl_autoload_register* (42) que permite registrar un método que debe ser invocado cada vez que se solicite alguna clase que no haya sido cargada anteriormente. Es en este momento donde el *framework* la carga antes de que PHP emita un error por no estar definida la clase solicitada. Estos tipos de métodos usualmente son llamados *autoload*.

Debido a que las clases pueden estar en cualquier fichero y un fichero puede tener la definición de más de una clase[7]; y además, estos ficheros pueden estar ubicado en cualquier directorio *lib* o en algunos de los subdirectorios, symfony guarda en caché la dirección completa de cada una de las clases del proyecto, de esta manera la carga de clases es más eficiente. El proceso de llenado de la caché se hace siempre la primera vez que se ejecuta la aplicación.

Para que el *plugin* cumpla con su objetivo se debe lograr modificar el mecanismo de carga automática del *framework*, garantizando que se carguen las clases modificadas con los *aspectos* y no las clases originales.

La función *spl_autoload_register* permite registrar más de una función *autoload* para la carga de clases y PHP las irá invocando en el mismo orden en que fueron registradas. El funcionamiento básico es el siguiente: cada vez que se solicite alguna clase que aún no ha sido cargada, PHP invoca a la primera función registrada pasándole por parámetro el nombre de la clase. Si después de ejecutado el método la clase sigue sin ser cargada, PHP invoca al segundo método en la cola y así sucesivamente hasta que alguna de las funciones cargue la clase. Si después que se acabe la lista de funciones registradas, la clase sigue sin ser cargada, PHP emitirá un error informando que la clase no ha sido definida. Después de esto, se puede concluir que para solucionar el problema basta con registrar una función *autoload* antes de que se registre la función propia del *framework*.

Todos los *plugin* pueden tener una clase *configuración* ubicada en el directorio *config*, y el nombre debe seguir la siguiente nomenclatura <*nombre_plugin*>*Configuration*, en este caso se llamaría *sfAspectPluginConfiguration*. Además, debe estar en un fichero que tenga como nombre, el nombre de la clase concatenado con *.class.php*, por lo tanto, en este caso sería *sfAspectPluginConfiguration.class.php* y por último la clase debe heredar de la clase *sfPluginConfiguration*.

La clase *sfPluginConfiguration* define tres métodos virtuales que pueden ser sobre escrito por las clases hijas para ejecutar código en distintos puntos de la ejecución del *framework*. La primera es el método *initialize* que es invocado cuando todo el proceso de inicialización ha concluido y es utilizado para hacer las configuraciones finales de los *plugin*. Los otros dos son los métodos *setup* y *configure* que tienen el mismo impacto, pues los dos son invocados uno a continuación del otro. Ambos son ejecutados prácticamente al inicio de la ejecución del *framework* cuando se comienza con la inicialización de la configuración del proyecto y lo que es más importante para los objetivos del *plugin*, antes de que se inicie la carga automática de Symfony.

El método *configure* fue el escogido para registrar la función *autoload* del *plugin*, pues registrarlo en este punto permite que entre a la cola de funciones *autoload* de PHP antes

[7] No es aconsejable implementar más de una clase por fichero ya que traería problemas a la hora de entender y gestionar el código. Teniendo una sola clase por fichero se podría poner como nombre del fichero el nombre de la clase.

que el *autoload* del *framework*. De esta manera se logra que esta función sea invocada siempre antes que la carga automática de symfony.

Cada vez que la función *autoload* del *plugin* es invocada, PHP le pasa por parámetro el nombre de la clase y es entonces que el *plugin* ejecuta los siguientes pasos:

1. Verifica si a la clase se le deben aplicar los *aspectos* (luego se verá la forma de indicarle al *plugin* a qué clase se le deben aplicar los *aspectos*).
2. Si se le deben aplicar los *aspectos*, entonces el *plugin* lanza el evento *aspect.ignore_class*. Este evento es de tipo *notifyUntil,* lo que significa que si algún *listener* responde *true*, entonces el *plugin* ignorará esta clase y no cargará los *aspectos*.
3. Si finalmente se deben cargar los *aspectos* para la clase, entonces el *plugin* verifica si ya el fichero de la clase modificada ha sido creado.
4. Si ya ha sido creado, se verifica si la fecha de la última modificación del fichero es mayor que la fecha de modificación de la clase original y de los ficheros de declaración de los *aspectos* a aplicar. De ser mayor, no se vuelve a generar la clase pues está actualizada.
5. Si no es mayor la fecha de modificación o aún no se ha creado el fichero de la clase modificada, entonces se genera el fichero y es guardado en la caché del proyecto para que la próxima vez el proceso sea más eficiente.
6. Luego de todo esto, a través del método *setClassPath* de la clase *sfAutoload* se modifica la dirección que tiene el *framework* de la clase original y se le pone la dirección de la clase modificada. A partir de ahí, el propio mecanismo de carga automática de symfony realiza el resto.

Como se mencionó anteriormente, las clases en un proyecto symfony pueden estar ubicadas en varias carpetas alrededor de todo el proyecto y generalmente no se les deben aplicar los *aspectos* a todas las clases o los *aspectos* a aplicar pueden variar en cada caso. Es necesario entonces configurar en qué directorio están las clases a las que se les aplicarán los *aspectos* y definir además, dichos *aspectos,* esto último se hace a través de ficheros XML.

La configuración de los *aspectos* se debe hacer en un directorio con el nombre *aspect*, ubicado en la carpeta de configuración del proyecto. Este comportamiento se puede modificar a través del fichero *config.yml* en el directorio *config* del *plugin*. El siguiente listado muestra su contenido por defecto:

Listado 5	Contenido por defecto de config.yml.

```
all:
  dir_aspect: %SF_CONFIG_DIR%/aspect
```

En la carpeta definida por *dir_aspect* deben estar ubicados todos los ficheros XML que contienen la definición de los *aspectos* (más adelante se verá un ejemplo de su definición). Además, debe tener un fichero con el nombre *config_aspect.yml* donde se definan los directorios en donde están las clases a las que se le deben aplicar los

aspectos. La definición de este fichero es similar al fichero *autoload.yml* de symfony. En el siguiente listado se pone un ejemplo de su uso:

Listado 6 Ejemplo del contenido del fichero config_aspect.yml.

```
aspect:
  project:              #Entrada de configuración, el nombre es a
                        #conveniencia del desarrollador, es solo
                        #una etiqueta.

    path: %SF_LIB_DIR%  #Camino donde están las clases a las cuales se
                        #les deben aplicar los aspectos.
                        #Para conocer la forma de darle valor a este
                        #campo se puede coger como muestra el fichero
                        #autoload.yml

    recursive: on       #Si está puesta en "on" todas las clases
                        #ubicadas en los subdirectorios serán
                        #procesadas y en "off" solo las clases
                        #que estén en el directorio raíz.

    exclude_dir: [model]#Nombre de los subdirectorios que deben ser
                        #excluidos. En este caso se está excluyendo al
                        #subdirectorio model. Si el arreglo está,
                        #ningún subdirectorio es excluido.

    exclude_class: []   #Nombre de las clases que debe ser excluidas,
                        #en este caso no se está excluyendo a ninguna
                        #clase.

    file_aspect: []     #Se ponen los nombres de los xml que contienen
                        #los aspectos que deben ser aplicados, de no
                        #ponerse ninguno, como en este caso, se
                        #aplicarán todos los aspectos definidos en los
                        #ficheros xml. De poner algún nombre de fichero
                        #no se debe poner la extensión xml.
```

En este fichero[8] se pueden definir tantas entradas como sean necesarias.

A continuación se mostrará el diagrama de clases de diseño realizado para la implementación del componente. Solamente se pondrán las clases principales, las otras pertenecen a la librería de clases "*Transparent PHP AOP*".

[8] Para conocer mejor el funcionamiento del fichero *config_aspect.yml* se puede revisar en la guía definitiva de symfony la forma en que se configura el fichero *autoload.yml,* pues este es similar.

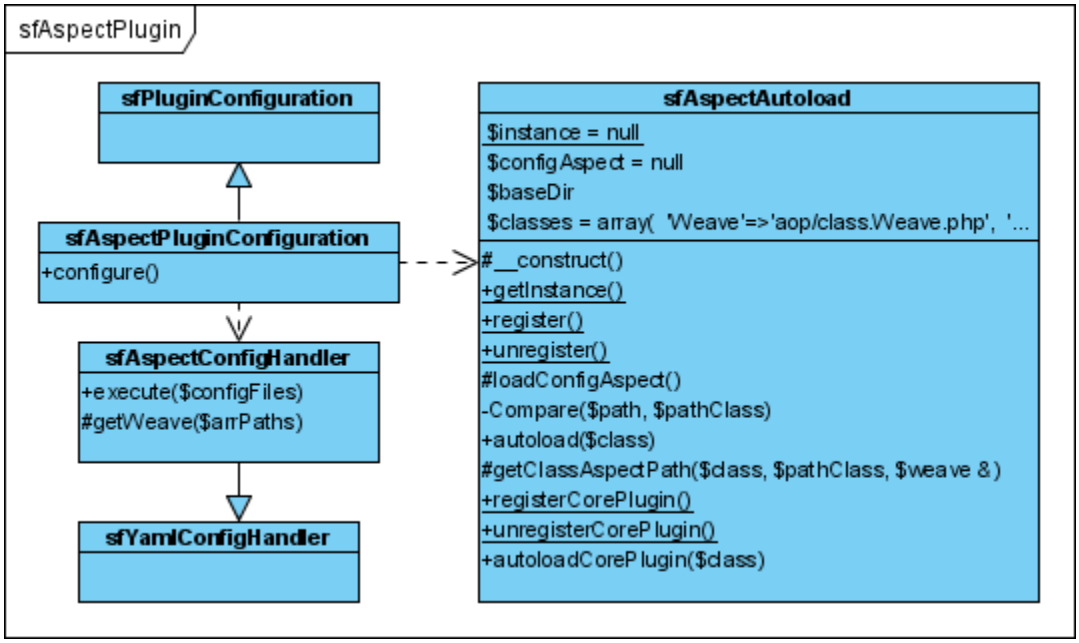


Figura: 6 Modelo de clases del diseño de *sfAspectPlugin.*

La clase *sfAspectPluginConfiguration* es la encargada de la configuración del *plugin*. Solo implementa el método *configure* como se explicó anteriormente. La clase *sfAspectConfigHandler* procesa el contenido del fichero *config_aspect.yml* y lo guarda en la caché del proyecto. Por último, la clase *sfAspectAutoload* es la que realiza el trabajo fundamental del *plugin*, aplicando los *aspectos* a las clases.

Integración de los componentes con el SAAA

El requerimiento no funcional de seguridad es uno de los primeros requerimientos que debe quedar resuelto en el diseño de un sistema. Debe ser tenido en cuenta desde que se concibe la línea base de la arquitectura. Este aspecto alcanza mayor importancia en las aplicaciones web, ya que este tipo de aplicaciones funciona sobre la red y pueden ser accedidas por cualquier usuario.

Symfony, a diferencia de otros *framework* para PHP, da soporte para implementar la seguridad en las aplicaciones. A través de una serie de ficheros de configuración se puede definir todo el modelo de seguridad de un sistema. A pesar de esto, no es posible gestionar de forma directa la seguridad en un componente del SISalud, debido a que ningún *framework*, por muy sofisticado que sea, se puede adaptar a las especificaciones de una arquitectura determinada. En esta sección se dará una explicación de cómo fue integrado el SAAA con el modelo de seguridad que propone symfony.

Modelo de seguridad de Symfony

Hacer una aplicación segura en symfony requiere de la realización de dos pasos: primero declarar los requerimientos de seguridad para cada acción en cada uno de los módulos y

segundo, autenticar a los usuarios con los privilegios correspondientes para poder acceder a las acciones (19).

En symfony cada vez que un usuario intenta ejecutar alguna acción en uno de los módulos, pasa por una serie de filtros. Uno de ellos es el filtro *security* en el cual se comprueban los requerimientos de seguridad de la acción y si el usuario tiene los privilegios suficientes para ejecutarla.

Los requerimientos de seguridad se definen en el fichero *security.ym* que está ubicado en el directorio *config* de cada uno de los módulos. Por cada una de las acciones se define si son seguras o no. En caso de ser seguras significa que los usuarios deben estar autenticados para poder ejecutarlas. Además, se definen las credenciales que debe poseer el usuario. Una credencial no es más que un nombre que permite organizar la seguridad por grupos.

Por defecto, las acciones son accesibles para todos los usuarios, por lo tanto, si no existe el fichero "*security.ym*" o no está definido el nivel de acceso para la acción solicitada, symfony no comprueba la seguridad y permite el libre acceso.

Cuando un usuario trata de acceder a una acción segura y restringida por credenciales ocurre lo siguiente:

- Si el usuario está autenticado y tiene las credenciales apropiadas, entonces la acción se ejecuta.
- Si el usuario no está autenticado, es redirigido a la acción de *login*.
- Si el usuario está autenticado pero no posee las credenciales apropiadas, será redirigido a la acción segura por defecto, que es una acción donde se le informa que ha intentado ejecutar una acción para la cual no tiene privilegios.

El mecanismo de seguridad de symfony se limita a dar acceso o no a un usuario según sus credenciales (rol) a los recursos de una aplicación. La forma de autenticar al usuario, el mecanismo que se escoja para clasificarlo en roles y el momento en que se va a ejecutar este proceso, corre por cuenta de los desarrolladores de los sistemas específicos.

Para symfony se encuentra disponible un *plugin* llamado *sfGuardPlugin* que contiene toda la lógica para el control de usuarios en una aplicación, pero a los desarrolladores del SISalud no les resulta útil, debido a que la forma de autenticar y clasificar a los usuarios no depende únicamente de las aplicaciones individuales, sino de su integración con el componente SAAA del SISaldud. Por lo tanto, se necesita desarrollar un mecanismo que permita lograr la integración con este componente.

Modelo de seguridad del SAAA

El SAAA es un componente más del SISalud, por lo tanto expone toda su funcionalidad a través de un servicio web. Este sistema es el encargado de la gestión de los usuarios y

los privilegios en las distintas aplicaciones y cada componente debe integrase a él para gestionar la seguridad. Además, permite guardar trazas de las operaciones realizadas por los usuarios en los sistemas.

Este componente posee dos métodos básicos, "*autenticar*" y "*listarderechos*". Ambos métodos retornan una estructura de datos llamada "credenciales de usuario" que contiene toda la información de los usuarios. Con esta información los sistemas pueden gestionar su seguridad. El SAAA en ambos métodos devuelve un objeto cuyo contenido se puede ver en el siguiente listado:

Listado 7 Contenido de las credenciales de usuario.

```
    [idusuario] => "id del usuario en el SAAA"
    [idusuarioexterno] => "id del usuario en el RC"
    [usuario] => "login del usuario"
    [nombre] => "nombre del usuario"
    [certificado] => "certificado del usuario, cadena de 32 caracteres"
    [nivel] => "nivel del usuario (nacional, provincial, municipal y
               unidad de salud)"
    [idnivel] => "id del la localización del usuario, según su nivel"
    [ListaDerechos] => Array
        (
            [0] => stdClass Object //por cada uno de los componentes
                                   //del SISalud
                (
                    [modulo] => "id del módulo en el SISalud"
                    [nombremodulo] => "nombre del módulo"
                    [tipousuario] => "tipo de usuario (rol)"
                    [url] => "url donde está ubicado el módulo"
                    [alias] => "nombre corto del módulo"
                    [descripcion] => "Descripción de los objetivos
                                      del módulo"
                    [imagen] => "nombre de la imagen que identifica
                                 al módulo"
                )
```

Para autenticar un usuario en uno de los componentes hace falta la información contenida en los elementos señalados en negrita:

- *ListaDerechos*: es un arreglo y contiene los requerimientos de seguridad en cada uno de los módulos.
- *modulo*: es el *id* del módulo al cual pertenece la información en cada posición del arreglo.
- *tipousuario*: es un entero que define el rol del usuario dentro del componente.

Por lo tanto, para autenticar un usuario basta con buscar si el componente es uno de los definidos en la lista de derechos, si está, entonces obtener el rol del usuario en ese componente.

Además, entre las credenciales de usuario hay un campo con el nombre "*certificado*", que es una cadena de 32 caracteres que se genera aleatoriamente en cada sección del usuario. Este dato es utilizado para el intercambio de información entre los componentes y se pone en la cabecera de los mensajes SOAP.

Llegado a este punto se encontraron dos problemas:

- ¿Cómo conocer el *id* del *componente* al cual quiere acceder el usuario?
- El tipo de usuario devuelto por el SAAA es un entero, sin embargo este tipo de dato no es conveniente utilizarlo dentro de la aplicación, debido a que symfony utiliza cadenas para definir las credenciales. El problema sería cómo lograr hacer la conversión entre un entero y una credencial de symfony.

Diseño propuesto

La solución propuesta crea una serie de clases que sirven de intermediarias entre la seguridad de symfony y el SAAA, garantizando que se cumplan los tres elementos del modelo AAA propuesto por el SISalud: la autenticación, el acceso y la auditoría. Un desarrollador con pocos conocimientos extras y con el estudio del modelo de seguridad de symfony puede configurar la seguridad de cualquiera de los componentes.

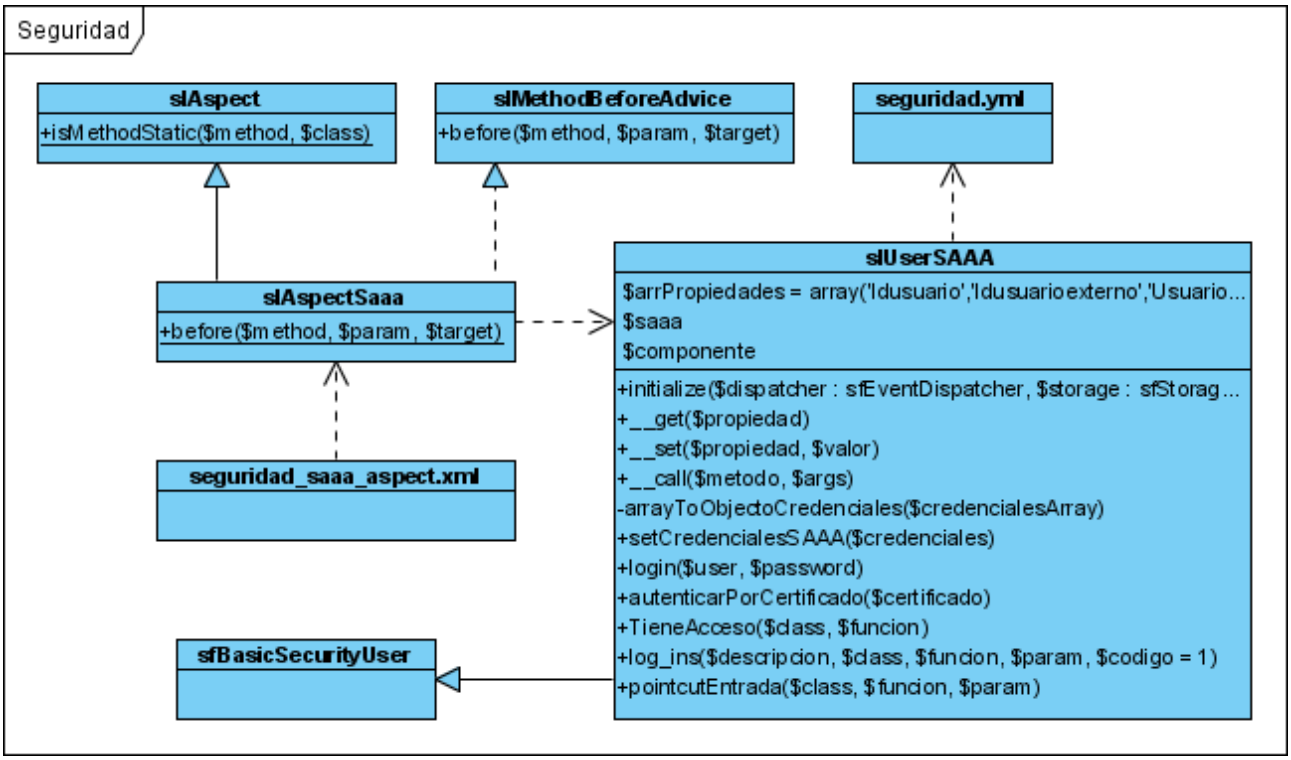


Figura: 7 Diagrama de clases del diseño de seguridad.

En symfony para la gestión de la seguridad, cuenta con una clase llamada *sfBasicSecurityUser,* la cual puede ser extendida para implementar la seguridad propia de los sistemas. Con este objetivo cada vez que se crea una aplicación[9], en el directorio *lib* aparece por defecto una clase con el nombre *myUser* que hereda de *sfBasicSecurityUser*. Los programadores deben programar en esta clase la seguridad propia de la aplicación.

Debido a que las aplicaciones del SISalud implementan la seguridad de la misma manera, se decidió implementar la clase *slUserSaaa,* que hereda de *sfBasicSecurityUser.* Esta clase también sirve de intermediaria entre los componentes y el SAAA, lo que permite disminuir el acoplamiento. Concentrar en la clase *slUserSaaa* toda la gestión de seguridad permite que sea más flexible modificar el modo de integración con el SAAA.

Symfony permite configurar qué clases van a ser utilizadas para la realización de las distintas tareas dentro de un proyecto, a través del fichero *factories.yml* ubicado en el directorio de configuración de las aplicaciones. Uno de los elementos que puede ser configurado, es la clase a utilizar en la gestión de la seguridad. En las aplicaciones del SISalud se debe poner que se va a utilizar la clase *slUserSaaa*. Esta forma de trabajo del symfony permite que si en algún momento se va a llevar el control de la seguridad a través de otro sistema, solo haría falta implementar una clase que sirva de intermediaria y configurar su uso en el fichero *factories.yml*.

Algunos de los métodos fundamentales de la clase son los siguientes:

- *setCredencialesSAAA*: guarda las credenciales del usuario en la sección para que puedan ser utilizadas luego en el sistema.
- *login*: autentica al usuario en el SAAA.
- *autenticarPorCertificado*: este método es útil a la hora de la creación de los WS, debido a que lo que se pasa en los mensajes SOAP es el certificado del usuario.
- *TieneAcceso*: dado el nombre de una clase y una función, se comprueba en el SAAA si el usuario tiene acceso para ejecutarla. Este método es útil para gestionar la seguridad en la lógica del negocio.
- *log_ins*: guarda trazas en el SAAA.
- *pointcutEntrada*: guarda una traza indicando la entrada de un usuario a un método determinado. Se utiliza a la hora de gestionar la auditoría.

Uno de los problemas que tiene la gestión de seguridad en el SISalud es que se debe buscar la manera de normalizar el modelo de seguridad de symfony y el del SAAA. Con este objetivo se creó el fichero de configuración *seguridad.yml,* que es utilizado por la clase *slUserSaaa* para realizar su trabajo. En el siguiente listado se muestra una explicación de su contenido.

Listado 8	Contenido del fichero seguridad.yml.

[9] Symfony utiliza una terminología propia para nombrar las partes de un sistema, diferente a la usada en la UCI, llama proyecto al sistema completo, aplicación a lo que en la universidad se le llama módulo y módulo a los casos de usos.

```
all:
  nombre_aplicacion: #Se pone una entrada como esta para cada
                     #aplicación

    modulo: 68 #id del Módulo en el SAAA.

    roles: #Tipos de usuarios del SAAA que tienen acceso al sistema.

      tipousuario_3: #por cada tipo de usuario se pone una entrada como
                     #esta. Se concatena la palabra "tipousuario" con
                     #el id en el SAAA

        credencial: medico_nefrologo #Se pone el nombre de la
                                     #credencial que se ve a usar
                                     #dentro de symfony

        titulo: Médico nefrólogo #Es un título legible para el usuario
                                 #final.
```

Para realizar la auditoría en los componentes se hace a través de la AOP. Específicamente se utiliza un solo *pointcut* que se aplicará al inicio de cada método en las clases del modelo, registrando una traza en el SAAA con la información del método y los parámetros que fueron pasados por el usuario. El fichero *seguridad_saaa_aspect.xml* contiene la definición de los aspectos.

La clase *slAspectSaaa* es la que realiza la implementación de la auditoría dentro de los sistemas. Hereda de la clase *slAspect* que tiene métodos útiles para la implementación del paradigma AOP. Además, la clase *slAspectSaaa* implementa la interfaz *slMethodBeforeAdvice* que define un método *before* donde se implementa el código que se desea ejecutar al inicio de cada método y es aquí donde se realiza la auditoría en los componentes.

En symfony es posible configurar el comportamiento de los sistemas para el caso de que un usuario intente ejecutar una acción para la que necesite estar autenticado y no lo esté. Además, se configura la acción segura por defecto a la cual va a ser redirigido cuando intente ejecutar una acción y no tenga los privilegios suficientes. Estos elementos de configuración se hacen en el fichero *settings.yml* de las aplicaciones.

El módulo *slSaaaAuth* perteneciente al *plugin* tiene las acciones *secure* y *login*. La primera utilizada como acción segura y la segunda para autenticar a los usuarios. En la siguiente figura se muestra el diagrama de clases del diseño.

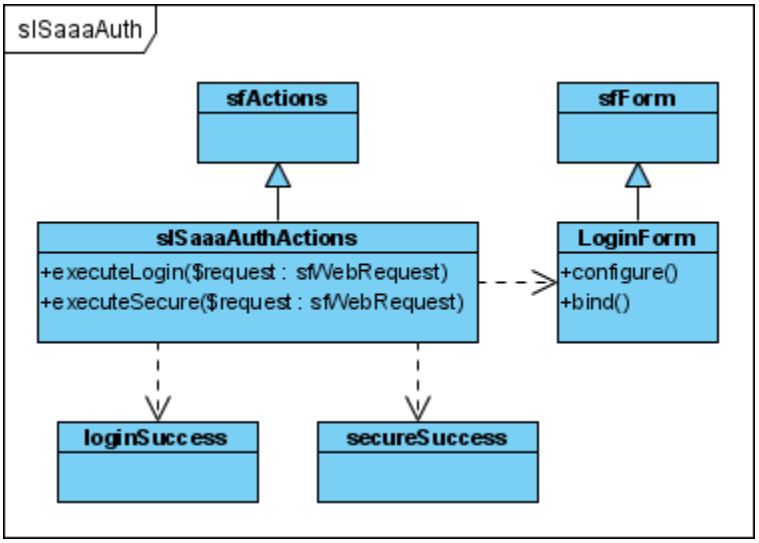


Figura: 8 Diagrama de clases del diseño del módulo slSaaaAuth.

La clase *slSaaaAuthActions* es la que implementa las acciones del módulo, y *loginSuccess* y *secureSuccess* son las plantillas de cada una respectivamente.

El peso de la lógica en este módulo lo lleva la clase *LoginForm* que es un formulario de symfony, por lo tanto hereda de la clase *sfForm*. En el formulario se definen dos *input,* uno para entrar el *login* del usuario y otro para el *password*. La clase redefine el método *bind* e invalida el formulario si no se puede autenticar el usuario.

Autenticar a los usuarios directamente en los sistemas introduciendo un *login* y una contraseña es útil cuando se están desarrollando los distintos componentes, debido a que a aun no han sido integrados al SISalud. Una vez integrados se debe garantizar "*Single Sign On*", por lo tanto este formulario verifica si lo que se ha pasado a la aplicación son las credenciales de los usuarios y automáticamente los autentica.

Gestión de eventos en el SISalud

En el año 2007 se implementó una nueva versión del sistema "Centro de Control", sistema que forma parte del SISalud y es el encargado de controlar el despliegue y la integración de los distintos componentes. Su objetivo es implementar una interfaz de usuario para trabajar con el SAAA, por lo tanto desde este sistema se pueden controlar además de los componentes, los usuarios y sus roles.

La implementación de esta versión tenía como uno de sus objetivos crear una plataforma que permitiera mantener la integridad referencial en la información distribuida en los distintos componentes. Para ello se implementó el patrón de diseño "Observador-Observable", configurándose las restricciones de integridad entre los sistemas, las cuales fueron clasificadas en dos tipos: *restrictivas* o en *cascada*.

A continuación se pondrá un ejemplo para ilustrar la necesidad de esta característica del SISalud. El Registro de Enfermedades de Declaración Obligatoria (REDO) homologa las

enfermedades de riesgo epidemiológico con el Registro de Clasificación Internacional de Enfermedades y Problemas Relacionados con la Salud (RCIE). Si se desea eliminar una determinada enfermedad en el componente RCIE, el cual tiene asociada una enfermedad de declaración obligatoria en el componente REDO y con el cual fue diagnosticado un paciente, traería consigo inconsistencia y corrupción en la información gestionada por REDO. Lo más lógico sería que se evitara la realización de esta acción, es decir, REDO es un observador restrictivo del componte RCIE (58).

Para la implementación del patrón, cada componente "*observador*" debe tener un método en su WS con el nombre *check*. Este método recibirá como parámetro la operación que se quiere ejecutar en el componte "*observado*" y los parámetros que se pasaron. La operación tiene la estructura: "*componente.método*". El tipo de dato que debe devolver el método es *booleano*, donde falso indica que se puede ejecutar la operación y verdadero en caso contrario.

Symfony tiene su propia implementación del patrón "Observador-Observable", que permite la gestión de eventos dentro de un sistema. Los eventos en el *framework* pueden ser lanzados y recibidos en cualquier parte de la aplicación, pero solo tienen efecto dentro de las aplicaciones individuales. Symfony no permite lanzar ni recibir eventos para y desde aplicaciones externas. La solución propuesta en este trabajo permite extender el modelo de eventos de symfony, lo que permite que dentro de las aplicaciones sea posible interactuar con el sistema de eventos del SISalud.

El identificador de un evento en symfony es una cadena de texto formada por un *namespace* y el nombre en sí del evento, unidos por un punto; y como se mencionó anteriormente el método *check* recibe como nombre de la operación una cadena con el siguiente formato "*componente.método*", un ejemplo sería "*RC.insertar_ciudadano*"[10]. Se decidió entonces que los eventos del SISalud cuando se ejecutan dentro de las aplicaciones reciban como nombre la cadena "SISalud_" concatenado con el nombre de la operación. Un ejemplo sería "*SISalud_RC.insertar_ciudadano*". Esto se hace con el objetivo de que el desarrollador tenga conciencia del tipo de evento del que se trata.

La siguiente figura muestra el diagrama de clases del diseño.

[10] Este es un método hipotético.

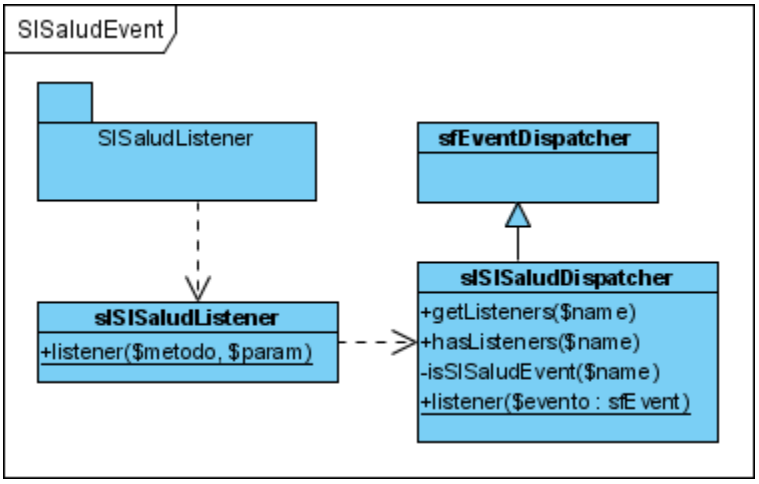


Figura: 9 Diagrama de clases del diseño de *SISaludEvent*.

Como el método *check* debe ser un método más del WS de la aplicación, se debe implementar un módulo que tenga una acción con este nombre. De esta manera cuando se genere el servicio web con el *ckWebServicePlugin* aparecerá en el WSDL. Se sugiere que este módulo tenga como nombre *SISaludListener*. La acción debe recibir como parámetro la operación y el arreglo de parámetros, luego debe invocar el método *listener* de la clase *slSISaludListener* para que haga el resto del trabajo. En el siguiente listado se muestra un ejemplo de cómo debe ser la clase *SISaludListenerActions*.

Listado 9 Ejemplo de la clase SISaludListenerActions.

```
<?php
class SISaludListenerActions extends sfActions
{
 /**
  *
  * @ws-enable
  * @ws-header Token: slTokenSaaa Este método es seguro.
  *
  * @param string $metodo Método del SISalud invocado
  * @param array $param  Parámetros pasado al método
  *
  * @return bool
  */
  public function executeCheck(sfWebRequest $request)
  {
      $metodo = $request->getParameter('metodo');
      $param  = $request->getParameter('param');

      $this->result = slSISaludListener::listener($metodo, $param);
  }
}
```

Para facilitar el desarrollo se implementó una tarea con el nombre *generate-listener*, que recibe como parámetro el nombre de una aplicación y automáticamente genera el módulo *SISaludListener* que contiene la acción *check*.

La clase *slSISaludListener* es la que recibe la operación y los parámetros que describen un evento del SISalud y lanza un evento dentro symfony para que sea procesado dentro de la aplicación, es decir, hace la conversión entre un evento del SISalud y un evento symfony. De esta manera se logra que los desarrolladores procesen este tipo de eventos como si fueran eventos normales de symfony.

El evento lanzado por *slSISaludListener* es de tipo *notifyUntil*, lo que garantiza que los *listener* puedan detener el procesamiento de un evento en la aplicación si devuelven verdadero después de procesarlo. Esto permite simular las relaciones de integridad de tipo "*restrictivas*" en el SISalud. La clase verifica si algún *listener* devolvió verdadero e informa que la operación del SISalud no puede ser ejecutada.

Por último, la clase *slSISaludDispatcher* hereda de *sfEventDispatcher* que pertenece al *framework* symfony. Esta clase es la utilizada para la gestión de eventos dentro de los componentes del SISaldud. Su objetivo es detectar cuando se lance un evento con el prefijo "*SISalud_*", debido a que debe ser lanzado fuera del sistema.

Actualmente el SISalud no permite que se lancen eventos desde los propios sistemas, es solo "Centro de Control" el que puede hacer esta tarea. No obstante, es interesante que los componentes individuales puedan lanzar este tipo de eventos, por lo que la incorporación de esta clase deja la posibilidad de que se pueda implementar esta característica en el futuro.

Modelo de excepciones

La gestión de errores dentro de una aplicación es uno de los puntos claves que debe ser analizado en el análisis y diseño. Se deben implementar mecanismos que permitan el reporte de errores desde cualquier parte del sistema y que estos contengan la descripción necesaria para que puedan ser tratados de forma eficiente. Con este objetivo PHP5 incorpora las excepciones, lo que permite la creación de sistemas robustos.

Dentro de *slSISaludPlugin* se crearon un grupo de clases para gestionar los distintos tipos de excepciones, las cuales se muestran en el siguiente diagrama de clases del diseño.

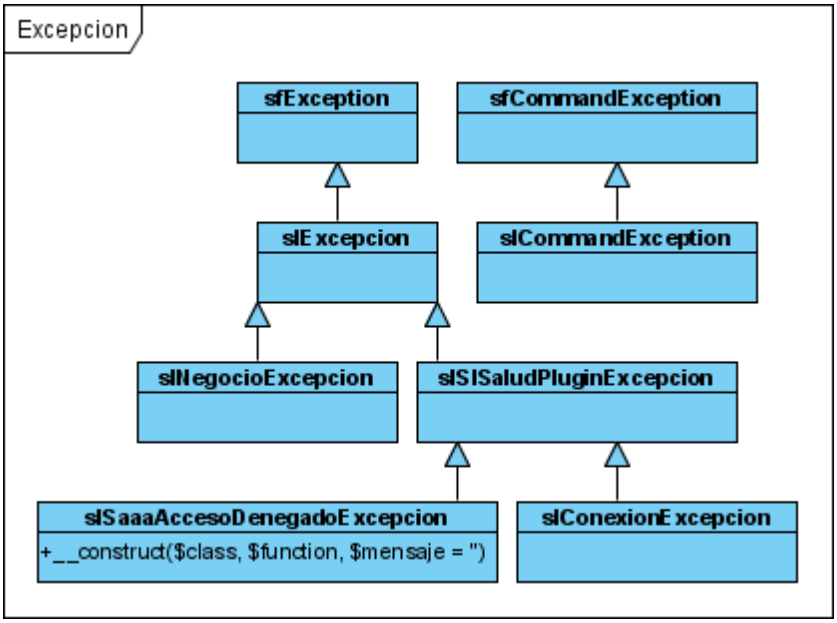


Figura: 10 Modelo de clases del diseño de excepciones.

A continuación una pequeña explicación de los objetivos de cada clase.

- *slCommandException*: es utilizada para reportar errores que se produzcan en la ejecución de un comando que se implemente para el *plugin*. Hereda de la clase *sfCommandException* que pertenece al symfony.
- *slExcepcion*: hereda de la clase *sfException* de symfony. Es la clase madre para todas las excepciones que se implementen para los componentes del SISalud.
- *slNegocioExcepcion*: todas las excepciones que se implementen para reportar errores del negocio de los componentes deben heredar de esta clase.
- *slSISaludPluginExcepcion*: clase madre para todas la excepciones que se implementen dentro del *puglin* vinculadas directamente con su funcionamiento.
- *slConexionExcepcion*: se utiliza para reportar errores en la integración entre los componentes. La utilización de esta clase permite la abstracción de los desarrolladores de la forma de conexión, debido a que todas las excepciones que se lanzan son de este tipo.
- *slSaaaAccesoDenegadoExcepcion*: para reportar error de acceso a los métodos del negocio, debido a que un usuario determinado no tiene los privilegios suficientes para ejecutarlo.

Gestión de trazas en el SISalud

La capacidad de guardar trazas sobre los sucesos que se producen en un sistema, permite entre otras cosas, la depuración y control de errores. De esta manera se pueden sacar reportes sobre los eventos más importantes ocurridos durante la ejecución de los programas.

Otras de las aplicaciones de las trazas es la de brindar mecanismos que permitan guardar la descripción sobre las acciones realizadas por los usuarios. A este proceso se le conoce

como auditoría y garantiza la responsabilidad de estos sobre los datos manejados por la aplicación. Cada operación realizada por el usuario es monitoreada y almacenada una descripción de la misma.

Con este objetivo, en el SAAA se implementó un método de su WS que permite el almacenamiento de trazas por parte de cualquier componente del SISalud. Esto permite que la gestión de la auditoría se haga de forma centralizada y homogénea. El método es "*log_ins*" y recibe como parámetro una descripción, un código, el componente que desea guardar el *log* y la acción realizada sobre el sistema.

Symfony también tiene su propio mecanismo de trazas que permite guardar *log* desde cualquier parte del sistema. Personalizar el mecanismo de *log* es muy sencillo, solo es necesario que las nuevas clases hereden "*sfLogger*" y que implementen el método "*doLog*" o que las nuevas clases implementen la interfaz "*sfLoggerInterface*" que define un método con el nombre "*log*".

La solución propuesta en este trabajo brinda mecanismos que permiten vincular los dos sistemas de trazas, siendo posible desde symfony guardar *log* directamente en el SAAA con los mecanismos propios que brinda este *framework*. Para esto se definió un nuevo nivel[11], al cual se puede acceder a través de una constante con el nombre "*SL_LOG*" definida en la clase "*slProdLogger*". En el siguiente listado se muestra un ejemplo donde se podrá ver la forma de guardar un *log* de este tipo.

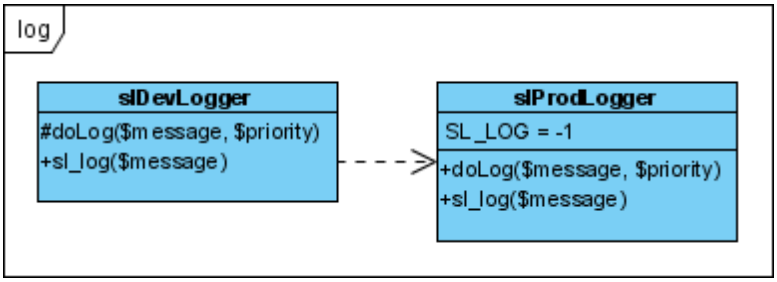
Listado 10 Forma de guardar una traza en el SAAA.

```
//Desde la acción
$this->logMessage($mensaje, slProdLogger::SL_LOG);

//Desde una plantilla
<?php use_helper('Debug') ?>
<?php log_message($mensaje, slProdLogger::SL_LOG) ?>

//Desde cualquier parte de la aplicación.
sfContext::getInstance()->getLogger()->sl_log($mensaje);
```

A continuación se muestra el diagrama de clases del diseño.



[11] En la guía definitiva de symfony se puede estudiar el mecanismo de *log* que propone el *framework*.

Figura: 11 Modelo de clases del sistema de log.

Cada componente del SISalud debe ser configurado para que en el entorno producción utilicen la clase "*slProdLogger*" en la gestión de log. Esta clase define un método con el nombre "*sl_log*" que es un forma rápida para guardar *log* directamente en el SAAA. En el listado anterior se vio un ejemplo de su uso. La clase hereda de "*sfLogger*" y redefine el método "*doLog*" y en el caso de que el nivel de un *log* coincida con la constante "*SL_LOG*" invoca el método "*log_ins*" de la clase slUserSaaa.

Por otra parte, la clase "*slDevLogger*" debe ser usada en el entorno de desarrollo. Hereda de la clase "*sfAggregateLogger*" e implementa el método "*doLog*". Su único objetivo es desechar las trazas que deben ser guardadas en el SAAA, ya que en desarrollo esto no es necesario.

Implementación

El componente *slSISaludPlugin* es un *plugin* para utilizarlo dentro de aplicaciones desarrolladas con symfony, por lo tanto su estructura es determinada por el *framework*. Contiene un directorio "*lib*" donde están los ficheros con la implementación de las clases o interfaces, el directorio "*config*" con los ficheros de configuración y el directorio "*modules*" donde son ubicados los módulos que pueden ser reutilizados en las distintas aplicaciones.

Symfony permite implementar más de una clase por fichero y todas sin importar su ubicación son beneficiadas por el mecanismo de carga automáticas del *framawork*, pero en el caso del *plugin* se siguió como estándar que cada clase e interfaz implementada debe ser definida en ficheros individuales. En el caso de las clases, el nombre del fichero se forma por la unión del nombre de la clase con "*.class.php*" y en el caso de las interfaces, por la unión del nombre de la interfaz con "*.interface.php*". Esta forma de trabajo permite que sea más sencillo entender la estructura del *plugin*. Además, las clases e interfaces con propósitos comunes fueron agrupadas en subdirectorios dentro de la carpeta "*lib*".

A continuación se presentan dos diagramas de paquetes, uno que muestra la estructura completa del *plugin* y otro con los elementos pertenecientes al directorio "*lib*".

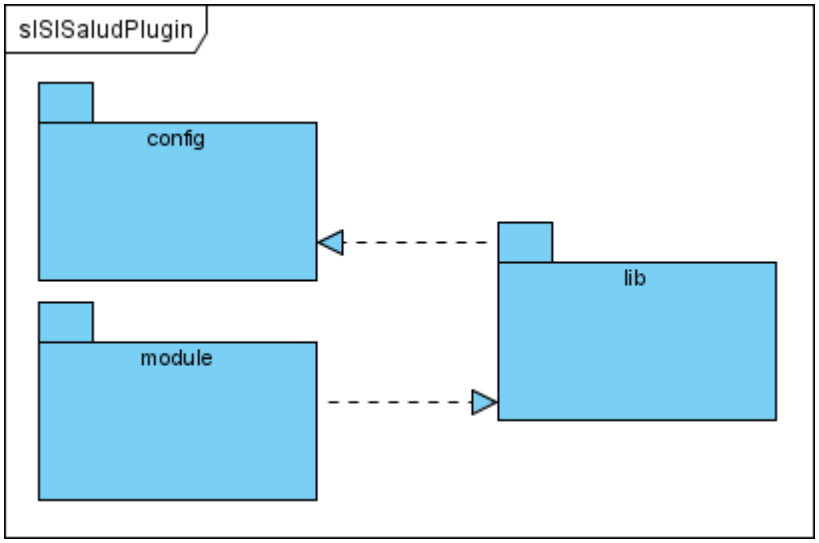


Figura: 12 Diagrama de paquetes de *slSISaludPlugin*.

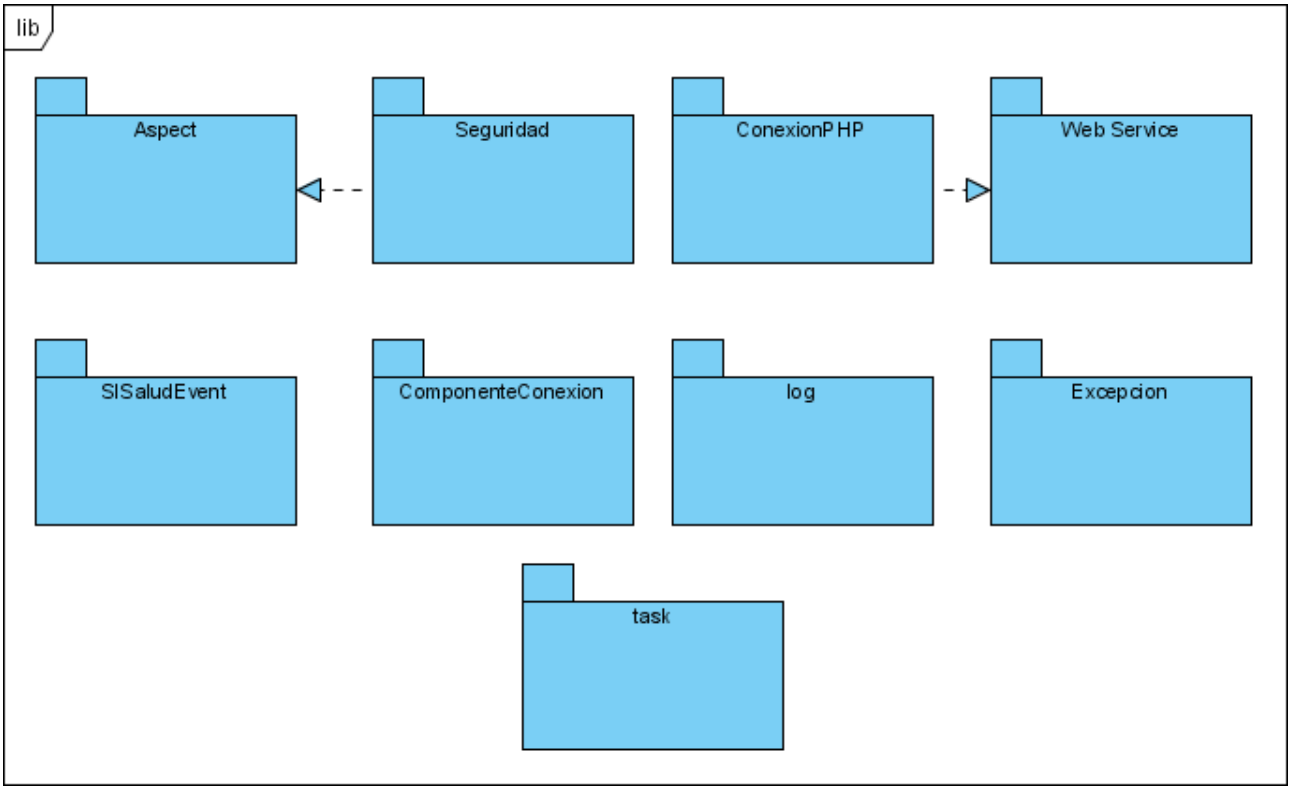


Figura: 13 Diagrama de paquete, *lib*.

En la siguiente tabla se muestra una breve explicación de cada uno de los paquetes y la relación de las clases que contiene.

Paquete.	**Descripción.**
Aspect	Las clases e interfaces utilizadas en la implementación del paradigma AOP. *slAspect*, *slMethodBeforeAdvice*
ComponenteConexion	Las clases utilizadas en la integración de los componentes. *slClassMap, slClientePHP, slClienteSoap, slCompFabrica,*

	slInterfaceComponente
ConexionPHP	Las clases para implementar, del lado del servidor, la integración de los componentes mediante PHP. *slConexionPHPController, slConexionPHPHandler.*
Excepcion	Las clases para el reporte de errores. *slCommandException, slConexionExcepcion, slExcepcion, slNegocioExcepcion, slSaaaAccesoDenegadoExcepcion, slSISaludPluginExcepcion*.
Log	Las clases para la gestión de las trazas. *slDevLogger, slProdLogger*
Seguridad	Las clases para la seguridad *slAspectSaaa, slUserSAAA*
SISaludEvent	Las clases para la gestión de eventos. *slSISaludDispatcher, slSISaludListener*.
task	En esta carpeta se implementan las tareas que ejecutará el *plugin* a través de líneas de comando. slGenerateListenerTask
WebService	Las clases para la implementación de los servicios web en los componentes. *slSoapHandler, slTokenHeaderListener, slTokenSaaa, slWebServiceController*.

Conclusiones del capítulo

Con la propuesta de solución –respecto al mecanismo de integración entre los componentes del SISalud para el intercambio de información-, se garantiza transparencia sobre el medio de comunicación que se utiliza en cada momento, de esta manera el programador se concentrará solo en resolver la lógica del negocio.

Por otro lado, aunque la utilización del *ckWebServicePlugin* permite la creación de forma sencilla de la interfaz de los servicios web de los componentes, fue necesaria la implementación de nuevos elementos que garantizara la seguridad del lado del servicio a través del tratamiento de la cabecera SOAP, pues PHP –a pesar de estar supuestamente implementado-, aun no procesa estas cabeceras.

Para la seguridad se desarrolló un conjunto de clases, que unido al *plugin sfAspectPlugin,* permite la implementación del modelo AAA en cada uno de los componentes, con lo cual, la integración con el SAAA queda garantizada y el desarrollador no deberá invertir esfuerzo en este sentido.

Finalmente, el *plugin* posee otro grupo de funcionalidades que permiten resolver un grupo de problemas puntuales en los sistemas, como es la gestión de eventos y trazas en el SISalud. Además de brindar un modelo de excepciones para reportes de errores.

Capítulo 3: Validación por Comité de Experto

Con el objetivo de validar la solución propuesta y de garantizar que cumple con los requerimientos necesarios para su utilización, se decidió evaluar el trabajo a través de un comité de expertos[12]. Estos fueron escogidos entre los desarrolladores con más experiencia en la implementación de aplicaciones para el SISalud, los cuales han tenido resultados comprobados en esta actividad.

Para la validación por el comité de experto, se escogió el método DELPHY, que es uno de los más utilizados con este objetivo. El método se utiliza para la realización de pronósticos con base subjetiva donde se utiliza la intuición como una comprensión sinóptica de la realidad y basados en la experiencia y conocimientos de un grupo de personas considerados expertos en la materia a tratar (59).

Después de haber pasado satisfactoriamente la validación por los expertos, se comenzó la utilización del *plugin* en el proyecto "Control Sanitario Internacional", perteneciente al "Departamento de Sistemas Especializados en Medicina", del "Centro Especializado en Soluciones de Informática Médica". Este proyecto cuenta con tres subsistemas: vectores, salud ambiental e higiene y epidemiología, los cuales tienen que interactuar entre ellos y con otros sistemas pertenecientes al SISalud. Aun este proyecto no ha concluido, pero el *plugin* ya ha demostrado su correcto funcionamiento.

Selección de los expertos

Para la selección de los expertos no se utilizaron muestreos probabilísticos, pues el propósito no era buscar representatividad en el sentido estadístico, sino garantizar juicios autorizados en el tema y por consiguiente validez en la información. Para ello se utilizó el muestreo intencional, que consiste en la selección de casos ricos en información, de los cuales se podían extraer conclusiones de gran relevancia para los propósitos del presente trabajo. Por tanto, se identificaron especialistas que llevaban como promedio tres años en el desarrollo de aplicaciones para el Sistema de Información para la Salud (SISalud), y con experiencias en el trabajo relacionado con la arquitectura de *software* utilizada[13].

Una vez que se les explicó a los expertos de manera individual en qué consistía el método que se aplicaría y después de haber dado su consentimiento para la evaluación, se les entregó un cuestionario impreso (ver anexo 2.1) que debían responder con el objetivo de valorar su nivel de competencia y conocimiento sobre el tema.

En esta metodología la competencia de los expertos se determina por el coeficiente **k**, el cual se calcula de acuerdo con la opinión del experto sobre su nivel de conocimiento

[12] Persona reconocida como una fuente confiable en un tema. Es un especialista con un elevado nivel de calificación, capaz de ofrecer valoraciones de un problema con un máximo de competencia.
[13] En el anexo 2.4 se puede ver una síntesis de sus currículos.

acerca del problema que se está resolviendo y con las fuentes que le permiten argumentar sus criterios.

El coeficiente de competencia se calcula por la siguiente fórmula:

K= ½ (kc + ka),

Donde:

kc: es el coeficiente de conocimiento o información que tiene el experto acerca del problema, calculado sobre la valoración del propio experto en una escala del 1 al 10, de esta forma, la evaluación "1" indica que el experto tiene muy poco conocimiento de la problemática correspondiente, mientras que la evaluación "10" significa todo lo contrario.

1	2	3	4	5	6	7	8	9	10

ka: es el coeficiente de argumentación o fundamentación de los criterios del experto, obtenido como resultado de la suma de los puntos alcanzados a partir de una tabla patrón como la siguiente:

FUENTES DE ARGUMENTACION	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis teóricos realizados por usted.			
Su experiencia obtenida.			
Trabajos de autores nacionales.			
Trabajos de autores extranjeros.			
Su propio conocimiento del estado del problema en el extranjero.			
Su intuición.			

Después del cálculo de **k,** se determina el nivel de competencia según la siguiente tabla:

	Coeficiente de competencia
0,7 < K < 1,0	Alto
0,4 < K < 0,7	Medio
K < 0,4	Bajo

En el procesamiento de esta ronda de información, se determinó que: los ocho expertos encuestados fueron seleccionados, pues presentaron un coeficiente de competencia entre 0,8 y 1,0.

Aplicación del método

En una segunda etapa, a los expertos seleccionados se les aplicó un segundo cuestionario (ver anexo 2.2) con el objetivo de que reflejaran cuáles eran los principales problemas enmarcados en el campo de acción del presente trabajo.

Con esto se comprobaría el nivel de coincidencia entre la opinión de los expertos y los elementos que se tuvieron en cuenta en la presente investigación respecto a los problemas existentes en las aplicaciones del SISalud. Ofrecería además, una medida de la necesidad real de dar solución a dichos problemas.

En otra etapa de la aplicación del método, se le aplicó al comité de experto una tercera encuesta (ver anexo 2.3), para evaluar finalmente la propuesta que se hace en el presente trabajo. Para responder a dicha encuesta se les entregó la documentación necesaria que explicaba en detalle cuáles era los objetivos del componente implementado.

La aplicación de estas encuestas se realizó de manera individual y bajo total anonimato, con el objetivo de que ningún miembro del equipo fuera influenciado por la reputación de otro o por el peso que supone la opinión de la mayoría. El experto puede defender sus argumentos tranquilamente sin el temor de que en caso de una equivocación, esta vaya a ser conocida por los otros. Esto posibilita obtener los verdaderos criterios de los participantes.

En esta encuesta se reflejan los criterios a evaluar en la propuesta, donde el experto debe responder si cada uno de esos criterios son: insuficientes, necesarios pero no suficientes o necesarios y suficientes. A parte de ello, el comité de experto debe responder otras preguntas que son realizadas con el propósito de valorar la utilidad e importancia de la propuesta.

Para el procesamiento de esta tercera ronda de información, se calculó la media aritmética en cada uno de los criterios a evaluar en cuanto al nivel de importancia de los mismos. Interpretando los resultados, “1” es de poco valor, y “10” es el máximo valor posible.

Resultados de la evaluación

Una vez procesada toda la información de la segunda encuesta, se obtuvieron los principales problemas –dentro de la arquitectura-, que afectan a todas las aplicaciones pertenecientes al SISalud, los cuales están relacionados a continuación junto con una breve explicación de cómo fueron resueltos en la propuesta de este trabajo.

1. Rigidez del sistema de seguridad SAAA (Sistema de Autenticación, Auditoria y Acceso): este punto no está en el alcance de este trabajo. La propuesta se limita solo a crear una infraestructura que garantice la integración de las aplicaciones al

sistema de seguridad del SISalud. No obstante, tener encapsulada toda la gestión de seguridad en la clase *slUserSAAA* permite poder cambiar a otro modelo de seguridad en el futuro, debido a que las aplicaciones no están directamente acopladas al SAAA.

2. La utilización del framework PLASER: en este caso, el trabajo sí incide directamente a este punto, permitiéndole a los desarrolladores implementar aplicaciones para el SISalud utilizando el *framework* symfony en lugar de PLASER.
3. Dificultades en los mecanismos de integración: la utilización del *slSISaludPlugin* unido al *ckWebServicePlugin* permite implementar de manera homogénea la integración entre los distintos componentes, utilizando tanto servicios web como directamente por PHP.
4. La arquitectura desde el inicio se definió orientada al lenguaje de programación PHP 4: una de las ventajas de utilizar *symfony* es que está desarrollado sobre PHP5, por lo que este punto es solucionado por esta vía y la integración entre las distintas aplicaciones sigue funcionando debido a que la comunicación es a través de servicios web.
5. La no utilización de Programación Orientada a Objeto: igual que en el punto anterior, la utilización de *symfony* y PHP5 permiten la programación orienta a objetos.
6. Alto acoplamiento entre los componentes: el acoplamiento entre las aplicaciones no depende en del *framework* utilizado, sino del negocio que gestione cada una de las aplicaciones. El nivel de acoplamiento depende del diseño particular de los componentes. No obstante, SISalud define una serie de restricciones en su arquitectura que no puede ser violada por los desarrolladores.
7. Única definición arquitectónica para todos los tipos de sistemas: la utilización de una arquitectura homogénea para todas las aplicaciones permite la reutilización del código y el conocimiento acumulado en su desarrollo. No obstante, debido a que la comunicación entre los distintos componentes es a través de servicios web cada aplicación es libre de utilizar la arquitectura y tecnología que satisfaga sus necesidades.
8. La no utilización de un ORM[14]: una de las ventajas de utilizar *symfony* es que viene integrado con un sistema ORM que puede ser Doctrine o Propel.

Estos resultados indican que la mayoría de los elementos que se tuvieron en cuenta para hacer la propuesta de solución en el presente trabajo, coincidieron con algunos de los elementos identificados por los expertos, como problemas en la arquitectura de los sistemas del SISalud.

Los resultados que arrojó la tercera encuesta fueron los siguientes:

[14] ORM: Siglas en inglés de “Mapeo de objeto relacional” es una filosofía que permite convertir datos entre el sistema de tipos de la programación orientada a objeto y el sistema de tipos de las bases de datos relacionales.

El 100 % de los expertos consideró que, la integración con el SAAA y la gestión de eventos en el SISalud, eran necesarios y suficientes para resolver los problemas planteados en este contexto.

El 87.5 % de los expertos consideró que la integración con otros componentes, la propuesta de implementación de la interfaz de servicio de los componentes y el modelo de excepciones, eran necesarios pero no suficientes. Estos resultados se reflejan en la siguiente tabla, donde: “I” es insuficiente, “NNS” es necesario pero no suficiente y “NS” es necesario y suficiente.

		Cantidades de expertos			**Por ciento (%)**		
No.	**Criterios a evaluar sobre la propuesta**	**I**	**NNS**	**NS**	**I**	**NNS**	**NS**
1	Integración con otros componentes.	0	1	7	0	12,5	87,5
2	Implementación de la interfaz de servicio de los componentes.	0	1	7	0	12.5	87,5
3	Integración con el SAAA.	0	0	8	0	0	100
4	Gestión de eventos en el SISalud.	0	0	8	0	0	100
5	Modelo de excepciones.	0	1	7	0	12.5	87,5

Otro de los resultados de esta tercera encuesta está relacionado con la media aritmética de cada uno de los criterios a evaluar.

En la siguiente tabla se refleja el grado de importancia, como promedio, que le conceden los expertos a cada uno de estos elementos.

	Cantidad de Expertos que evaluó cada criterio										
	Evaluaciones										
Criterio	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Media Aritmética**
1									1	7	9,875
2						1	1		2	4	8,875
3									1	7	9,875
4							2	3		3	8,5
5						1	2	1	2	2	8,25

Como se mencionó con anterioridad, los valores que están cercanos a “10” indican que esos criterios evaluados son de máxima importancia, mientras que los cercanos a “1” indican lo contrario. Por tanto, a partir de los resultados obtenidos, se puede comprobar que cada uno de los elementos que se evaluaron, fueron considerados por los expertos de gran relevancia, pues los valores de la media aritmética de cada uno de ellos, oscila entre 8,57 y 9,85.

Conclusiones del capítulo

Para la validación del presente trabajo se aplicó el método DELPHI, con el cual se obtuvieron resultados satisfactorios.

Esto demostró la importancia y necesidad de poner en marcha la aplicación de la propuesta presentada, en los sistemas que sean desarrollados para el SISalud y utilicen el *framework* symfony.

Para la validación del trabajo se seleccionó un grupo de expertos con bastante experiencia en el desarrollo de aplicaciones para el SISalud, que en su mayoría tenían entre 3 y 4 años de trabajo en esta actividad. Una gran parte de ellos son miembros del proyecto APS que es el más antiguo y el cual fue la base para la creación de los otros proyectos.

Se aplicó el método Delphi siguiendo la metodología que este plantea, lo que permitió la aplicación de las distintas encuestas y su procesamiento. Los resultados obtenidos fueron satisfactorios, demostrándose la validez de la propuesta de solución dada en este trabajo.

Después de este proceso se comenzó la utilización del slSISaludPlugin en los proyectos “Colaboración Internacional” y “Control Sanitario Internacional”. Ambas proyectos forman parte del SISalud, lo que ha permitido validar de forma práctica la solución propuesta.

Conclusiones generales

El desarrollo del *plugin* para symfony "*sfSISaluPlugin*" es una solución orientada a resolver un conjunto de problemas que fueron identificados en las aplicaciones que actualmente se desarrollan para el SISalud, como:

- Integración con otros componentes, capacidad que deben tener las aplicaciones para intercambiar información entre ellas.
- Integración con el SAAA, sistema que se encarga de la gestión de la seguridad dentro del SISalud. Es un requerimiento del grupo de arquitectura de que cada una de las aplicaciones lleve el control de seguridad a través de este sistema.
- Modelo de eventos. En el trabajo se muestra la manera de integrar el sistema de eventos del symfony con el modelo de eventos del SISalud, que le permite a este último controlar la integridad referencial entre los sistemas.
- Modelo de excepciones. Se implementa un conjunto de clases para el trabajo de excepciones que permite la construcción de aplicaciones más robustas.

La utilización del *plugin* permitirá la creación de una arquitectura homogénea para todos los sistemas del SISalud, de esta manera lo único que cambiaría es el negocio particular que cada uno gestiona. Aumentará la reutilización de código y el conocimiento que se genere en los distintos grupos de desarrollo, lo que permitirá disminuir el tiempo de desarrollo y la estabilidad de los componentes.

La integración entre los componentes que utilicen el *plugin* podrá ser implementada a través de la utilización de diferentes vías de comunicación y no solo usando servicios web como se hace actualmente. Con este objetivo se utilizó una arquitectura flexible, lo que permitió el desarrollo de la integración directamente por PHP para aquellas aplicaciones que se ejecuten en el mismo servidor.

La gestión de la seguridad es un punto crucial en el desarrollo de los componentes. La seguridad se implementa a través de la integración con el SAAA, garantizando la auditoría, la autenticación y el acceso. La solución permite combinar la seguridad propuesta por el *framework* symfony y el modelo de seguridad del SISalud. Específicamente para la realización de la auditoría se utilizó el paradigma de la Programación Orientada a Aspecto a través de la implementación del componente *sfAspectPlugin*.

El trabajo fue validado por un comité de expertos –utilizando el método Delphi-, los cuales fueron escogidos entre los más experimentados dentro de la comunidad de desarrollo de aplicaciones para el SISalud. Además, se comenzó la utilización del *plugin* en los proyectos "Control Sanitario Internacional" y "Colaboración Internacional", lo que ha permitido validar de forma práctica su funcionamiento.

Recomendaciones

Después de haber realizado la propuesta de solución para algunos de los problemas que afectan el desarrollo de aplicaciones para el SISalud y teniendo en cuenta las consultas realizadas a un conjunto de expertos en el tema, se identificaron algunos aspectos que deben ser tratados en próximas investigaciones, los cuales se listan a continuación:

- Implementación de componentes que permitan el desarrollo de las interfaces de los sistemas. Se sugiere la utilización de algún *framework* para el trabajo con *ajax*.
- Continuar el desarrollo del trabajo con la programación orientada a aspecto utilizando el lenguaje PHP.
- Integrar alguna herramienta para la generación de reportes.
- Creación de una herramienta que permita la gestión de los WSDL de los distintos componentes.
- Desarrollo de un modelo de seguridad que sustituya al actual SAAA.
- Investigar sobre la utilización de un ESB (*Entreprise Service Bus*).
- Utilización de algún estándar de comunicación entre aplicaciones médicas como HL7.

Glosario de Término y Siglas

- AAA: Autenticación, Autorización y Auditoría.
- AJAX: Acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML) es una técnica de aplicaciones web para crear aplicaciones interactivas. Este tipo de aplicaciones se ejecuta en el cliente, es decir, en el navegador del usuario, manteniendo la comunicación asíncrona con el servidor en segundo plano.
- AOP: Siglas en inglés de *Aspect Oriented Programming* que significa Programación Orientada a Aspecto.
- API: (Application programming interface) Es una interfaz de programa de software que se implementa para que otro software pueda interactuar con él. Usualmente las API son implementadas por librerías, aplicaciones y sistemas operativos, donde define como otros programas pueden acceder a distintos servicios que se brindan.
- APS: Atención Primaria de Salud, nombre que se le dio a uno de los primeros proyectos que comenzó el proceso de informatización del Sistema Nacional de Salud.
- COP: *Component Oriented Programming*. Programación Orientada a Componentes.
- CUS: Casos de uso del sistema.
- ECS: Elementos de Configuración de Software.
- MINSAP: Ministerio de Salud Pública.
- OASIS: Acrónimo de (*Organization for the Advancement of Structured Information Standards*). Es un consorcio internacional sin fines de lucro que orienta el desarrollo, la convergencia y la adopción de los estándares de comercio electrónico y servicios web.
- ORM: O*bject-Relational Mapping*. Mapeo de objeto-relacional.
- POO: Programación Orientada a Objetos.
- RC: Registro de Ciudadano.
- RIS: Registro Informatizado de Salud.
- SAAA: Sistema de Autenticación Auditoría y Acceso. Sistema que forma parte el SISalud como uno de sus componentes y es el encargado de la gestión de usuarios en todos los sistemas.
- SISalud: Sistema de Información para la Salud.
- SNS: Sistema Nacional de Salud.
- SOAP: Siglas de "*Simple Object Access Protocoles*", un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Es uno de los protocolos utilizados en los servicios Web.
- TIC: Tecnologías de la Información y las Comunicaciones.
- UCI: Universidad de las Ciencias Informáticas.

- WS: Siglas de “*Web Service*”, que es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones, a través del uso de XML.
- WSDL: son las siglas de “*Web Services Description Language*”, un formato XML que se utiliza para describir servicios web.
- XSLT: Lenguaje declarativo basado en XML, que permite la transformación de documentos XML en otro documento XML.
- XML (Extensible Markup Language) Lenguaje de marcado extensible. Es un grupo de reglas para codificar documentos electrónicos.

Trabajos citados

1. **Hernández, Mirna Cabrera, Ramos, Ariel Delgado y Rodríguez, Alfredo Sánchez.** PLATAFORMA PARA LA ADMINISTRACIÓN, PROCESAMIENTO Y TRANSMISIÓN DE LA INFORMACIÓN EN EL SISTEMA DE SALUD: SISALUD. [En línea] 2009. [Citado el: 30 de mayo de 2009.] http://informatica2009.sld.cu/Members/mirnacabrera/plataforma-para-la-administracion-procesamiento-y-transmision-de-la-informacion-en-el-sistema-de-salud-sisalud/at_download/trabajo.

2. La informatización en Cuba. *Cuba Minrex.* [En línea] 16 de noviembre de 2005. [Citado el: 30 de Mayo de 2009.] http://www.cubaminrex.cu/Sociedad_Informacion/Cuba_SI/Informatizacion.htm.

3. **Corales, Yosvanis Sánchez, y otros, y otros.** ENTORNO DE DESARROLLO PARA EL PROYECTO SISALUD. *Informática en salud.* [En línea] 2009. [Citado el: 29 de Mayo de 2009.] http://informatica2009.sld.cu/Members/mirnacabrera/entorno-de-desarrollo-para-el-proyecto-sisalud-edsisalud/at_download/trabajo.

4. **López, María del Carmen Paderni, Vitón, Caridad Guzmán y Drago, Daniel Dieppa.** NOMENCLADORES NACIONALES DE RECURSOS Y SERVICIOS PARA LA INFORMATIZACIÓN DE LA ATENCIÓN MÉDICA EN EL SISTEMA NACIONAL DE SALUD. *Informatica en salud 2009.* [En línea] 2009. [Citado el: 29 de Mayo de 2009.] http://informatica2009.sld.cu/Members/denis/nomencladores-nacionales-de-recursos-y-servicios-para-la-informatizacion-de-la-atencion-medica-en-el-sistema-nacional-de-salud/at_download/trabajo.

5. **Ramos, Ariel Delgado y Ledo, María Vidal.** Informática en la salud pública cubana. *Rev Cubana Salud Pública.* [En línea] 2006. [Citado el: 29 de Mayo de 2009.] http://bvs.sld.cu/revistas/spu/vol32_3_06/spu15306.htm#cargo.

6. **Velázquez, Karel Gómez.** *DOCUMENTO DE ARQUITECTURA DE SOFTWARE FACULTAD 7 Versión 1.2.* 2008.

7. **Fallside, David y Lafon, Yves.** W3C. *XML Protocol Working Group Charter.* [En línea] 2002. [Citado el: 3 de junio de 2009.] http://www.w3.org/2000/09/XML-Protocol-Charter.

8. **Computos, catedra: Gestión de Centros de.** *UNIVERSO SOA (Service Oriented Architecture).* [En línea] [Citado el: 16 de 3 de 2010.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=4&ved=0CA0QFjAD&url=http%3A%2F%2Fgestion.zobyhost.com%2Findex.php%3Foption%3Dcom_rubberdoc%26view%3Ddoc%26id%3D19%26format%3Draw&rct=j&q=SOA+surgimiento+necesidad&ei=crafS9LoNsX6lwffppWYDg&usg=AFQjCNH.

9. **MacKenzie, C. Matthew.** Reference Model for Service Oriented Architecture 1.0. *OASIS Standard.* [En línea] octubre de 2006. [Citado el: 3 de 6 de 2009.] http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

10. **Reynoso, Billy.** Arquitectura Orientada a Servicios (SOA). [En línea] [Citado el: 4 de 6 de 2009.] http://download.microsoft.com/download/4/F/F/4FF88340-43CC-4C5B-8E50-09002969D0DD/20051129-ARC-BA.ppt.

11. **Katrib, Miguel, y otros, y otros.** *VisualStudio.Net 2008 desafía todos los retos.* Ciudad de la Habana, Cuba : Capitán San Luis, 2008. 978-959-211-329-9.

12. **Julián Astorga Campos, Juan Luis Quirós Venegas.** UN ENFOQUE TEÓRICO E INTUITIVO A LA ARQUITECTURA ORIENTADA A SERVICIOS (SOA). [En línea] [Citado el: 15 de 3 de 2010.] http://www.di-mare.com/adolfo/cursos/2007-2/pp-SOA.pdf.

13. **Carlos Reynoso, Nicolás Kicillof.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [En línea] http://bibliodoc.uci.cu/pdf/estilos.pdf.

14. **Aires, Departamento de Computación Facultad de Ciencias Exactas Universidad de Buenos.** *Tendencias Tecnológicas en Arquitecturas y Desarrollo de Aplicaciones.* [En línea] [Citado el: 17 de 3 de 2010.] http://www-2.dc.uba.ar/materias/nt/ttada-2004.pdf.

15. **Camargo, Jorge.** Arquitectura de Software: Caso práctico. [En línea] [Citado el: 4 de 6 de 2009.] http://camargoj.googlepages.com/ccc2007-arquitectura-final.pdf.

16. **Jonás A. Montilva C., Nelson Arapé y Juan Andrés Colmenares.** *Desarrollo de Software Basado en Componentes.* [En línea] [Citado el: 13 de 3 de 3010.] http://webdelprofesor.ula.ve/ingenieria/jonas/Productos/Publicaciones/Congresos/CAC03%20Desarrollo%20de%20componentes.pdf.

17. **A. Suárez, J. L. De Armas, T. Aznielle.** *Modelo de una familia de equipos de neurofisilogía a través de componentes visuales* . [En línea] [Citado el: 17 de 3 de 2010.] http://www.hab2005.sld.cu/arrepdf/T061.PDF.

18. **Jorge Fontenla González, Manuel Caeiro Rodríguez, Martín Llamas Nistal.** *Una Arquitectura SOA para sistemas de e-Learning a través de la integración de Web Services.* [En línea] [Citado el: 17 de 3 de 2010.] http://www.ieec.uned.es/Investigacion/RedOber/archivos/Cita2009%20art4%20redober.pdf.
19. **Fabien Potencier, François Zaninotto.** *Symfony la guía definitiva.* París : s.n., 2008.
20. **Team, Symfony.** symfony. *Open-Source PHP Web Framework.* [En línea] [Citado el: 19 de 3 de 2010.] http://www.symfony-project.org/.
21. **Team, Google.** Comparacion de framework. *Google trends.* [En línea] http://google.com/trends?q=symfony%2C+Zend+Framework%2C+CakePHP%2C+codeigniter&ctab=0&geo=all&date=2009&sort=0.
22. **Adelaide Bianchini, Ricardo Blanch, Maruja Ortega, Ascánder Suárez.** DISEÑO DE UNA HERRAMIENTA PARA EL DESARROLLO DE APLICACIONES WEB BASADAS EN STRUTS. [En línea] [Citado el: 23 de 3 de 2010.] http://www.iadis.net/dl/final_uploads/200713L012.pdf.
23. **Leisalu, Oliver.** Comparative Evaluation of Two PHP Persistence Frameworks. [En línea] [Citado el: 23 de 3 de 2010.] http://sep.cs.ut.ee/uploads/Main/LeisaluBachelorThesis.pdf.
24. **François Zaninotto.** Comparing Propel, Doctrine and sfPropelFinder . *Redo The Web.* [En línea] [Citado el: 24 de 3 de 2010.] http://redotheweb.com/2008/07/08/comparing-propel-doctrine-and-sfpropelfinder/.
25. **Potencier, Fabien.** Doctrine vs Propel. *symfony team.* [En línea] [Citado el: 24 de 3 de 2010.] http://www.symfony-project.org/blog/2009/12/07/doctrine-vs-propel.
26. **Group, The PHP.** PEAR Manual. *Pear.* [En línea] [Citado el: 6 de 4 de 2010.] http://pear.php.net/manual/en/.
27. **Caridad Guzmán Vitón, Denis Derivet Thaureaux, Lucía E. Domínguez Abreu,Yamilka Gómez León, Yosvani Turruelles Tejeda, Danieski Rodríguez Osoria,Alexander Paneque Meschenkov.** NOMENCLADORES NACIONALES GEOGRÁFICOS PARA LA INFORMATIZACIÓN DE LA ATENCIÓN MÉDICA EN EL SISTEMA NACIONAL DE SALUD . [En línea] [Citado el: 30 de 3 de 2010.] http://www.informatica2009.sld.cu/Members/denis/nomencladores-geograficos-nacionales-para-la-informatizacion-de-la-atencion-medica-en-el-sistema-nacional-de-salud-1/at_download/trabajo.
28. **Rodríguez, Alfredo Sánchez y Sourd, Frank Pompa.** *Normas para el desarrollo de aplicaciones para la salud en Cuba.* C. Habana : s.n., 2007.
29. **Isaza, Pablo Figueroa y Alejandro.** Programación por Aspectos. Una Introducción. [En línea] 9 de mayo de 2006. [Citado el: 25 de mayo de 2009.] http://agamenon.uniandes.edu.co/~pfiguero/Papers/poa-borrador.pdf.
30. **Quintero, Antonia Mª Reina.** Visión General de la Programación Orientada a Aspectos. [En línea] Dicimbre de 2000. [Citado el: 25 de mayo de 2009.] http://www.lsi.us.es/docs/informes/aopv3.pdf.
31. **Ann L. Winblad, Samuel D. Edwards, David R. King.** Object Oriented Software. [En línea] [Citado el: 29 de 3 de 2010.] http://www.google.com/books?hl=es&lr=&id=aYSqlVwImrQC&oi=fnd&pg=PR11&dq=Programacion+orientada+a+objetos&ots=4YfpEQVXeL&sig=2H2j4fuC9BqGasHXW00CE5EExmk#v=onepage&q=&f=false.
32. **Fernando Asteasuain, Bernardo Ezequiel Contreras.** PROGRAMACIÓN ORIENTADA A ASPECTOS, pag 6. [En línea] [Citado el: 31 de 3 de 2010.] http://www.angelfire.com/ri2/aspectos/Tesis/tesis.pdf.
33. **Ulises Juárez Martínez, José Oscar Olmedo Aguirre.** Entorno dinámico para mejorar la expresividad de los lenguajes orientados a aspectos. [En línea] [Citado el: 31 de 3 de 2010.] http://computacion.cs.cinvestav.mx/~ujuarez/aniei_20040715.pdf.
34. **Fernando Asteasuain, Leandro Ariel Schmidt.** Aplicación de la Programación Orientada a Aspectos como Solución a los Problemas de la Seguridad en el Software. [En línea] [Citado el: 31 de 3 de 2010.] http://www.angelfire.com/ri2/aspectos/Tesis/Final.pdf.
35. **Javier J. Gutiérrez, Darío Villadiego, María J. Escalona, Manuel Mejías.** APLICACIÓN DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS EN EL DISEÑO E IMPLEMENTACIÓN DE PRUEBAS FUNCIONALES. [En línea] [Citado el: 31 de 3 de 2010.] http://quercusseg.unex.es/juanmamu/DSOA04/papers/gutierrez-lladiego-escalona-mejias.doc.
36. **Bergmann, Sebastian.** AOP @ PHP The State of Aspect-Oriented Programming for PHP. [En línea] Septiembre de 2006. [Citado el: 5 de mayo de 2009.] http://sebastian-bergmann.de/talks/2006-09-14-AOP.pdf.
37. **Garcia, Jorge Esparteiro.** Aspect-Oriented Web Development in PHP. [En línea] [Citado el: 2 de 4 de 2010.] http://paginas.fe.up.pt/~prodei/DSIE08/papers/46.pdf.
38. **Sebastian Bergmann, Günter Kniesel.** GAP: Generic Aspects for PHP. [En línea] [Citado el: 2 de 4 de 2010.] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.3637&rep=rep1&type=pdf.

39. **Pressman, Roger S.** *Ingeniería de Software, un enfoque práctico.*
40. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software.*
41. **Andrés Castillo, Oscar Sanjuán, Luis Joyanes.** Aportación del paradigma orientado a agentes en el desarrollo de tutores de e-learning. [En línea] [Citado el: 31 de 3 de 3010.] http://www.dccia.ua.es/jenui2004/actas/ponencias/ponencia31.pdf.
42. **Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy, Jakub Vrana.** Manual de PHP. [En línea] 2009. http://www.php.net/manual/es/.
43. **Corporation, Microsoft.** La Arquitectura Orientada a Servicios (SOA) de Microsoft. *Microsoft SOA & Business Process.* [En línea] diciembre de 2006. [Citado el: 8 de enero de 2010.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=3&ved=0CA4QFjAC&url=http%3A%2F%2Fdownload.microsoft.com%2Fdownload%2Fc%2F2%2Fc%2Fc2ce8a3a-b4df-4a12-ba18-7e050aef3364%2F070717-Real_World_SOA.pdf&rct=j&q=SOA&ei=2hyVS5a3EIjilweE8tz7AQ&usg=AFQjCNEiHNL4.
44. **José Luis Isla Montes, Francisco Luis Gutierrez Vela.** Modelo Estructural de Patrones de Diseño: Diagramas REP. [En línea] [Citado el: 2 de 4 de 2010.] http://www.willydev.net/InsiteCreation/v1.0/descargas/willydev_joseluis_diagramas_rep.pdf.
45. **Campo, Gustavo Damián.** Patrones de Diseño, Refactorización y Antipatrones. Ventajas y Desventajas de su Utilización en el Software Orientado a Objetos. [En línea] [Citado el: 2 de 4 de 2010.] http://www.ucasal.net/templates/unid-academicas/ingenieria/apps/4-p101-Campo.pdf.
46. **Jose B. García Perez-Schofield1, Emilio García Roselló.** Notas sobre la construcción de simulaciones y su integración desde un punto de vista orientado a objetos. [En línea] [Citado el: 2 de 4 de 2010.] http://webs.uvigo.es/grupoimo/articulos/OOP-CONIED99%20.PDF.
47. **Oscar A. León, Mariana Brachetta, Julio Monetti.** HERENCIA MÚLTIPLE EN JAVA. [En línea] [Citado el: 2 de 4 de 2010.] http://dc.exa.unrc.edu.ar/nuevodc/materias/analisis/rep/1192566190/Herencia%20Multiple%20en%20Java.pdf.
48. **Mercado, Carlos Argelio Arévalo.** USO DE METRICAS DE SOFTWARE EN EL CONTEXTO DEL DEBATE DE LA ENSEÑANZA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS COMO PRIMER PARADIGMA. [En línea] [Citado el: 2 de 4 de 2010.] http://www.sig-ed.org/ICIER2007/proceedings/uso_de.pdf.
49. **Quintero, Carlos Enrique Cuesta.** Ingeniería de Software II Tema 2: Principios del Diseño del Software. [En línea] [Citado el: 2 de 4 de 2010.] http://kybele.escet.urjc.es/documentos/IS2/IS2-2_Segunda_Parte.pdf.
50. **Vázquez, Jacinto Mata.** PÁGINAS WEB DINÁMICAS CON EL LENGUAJE DE SCRIPTS PHP. pags. 53-57. *Sólo programadores.* 70, 2000, ISSN 1134-4792.
51. **Team, PHP.** runkit. [En línea] [Citado el: 2 de 4 de 2010.] http://www.php.net/runkit.
52. **Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana.** Web Services Description Language (WSDL) 1.1. *w3c.* [En línea] [Citado el: 2 de 4 de 2010.] http://www.w3.org/TR/wsdl.
53. **Daniel Dieppa Drago, Alfredo Sánchez Rodríguez Otniel Barrera Palenzuela.** GESTIÓN DE LA ARQUITECTURA DEL PROYECTO DE INFORMATIZACIÓN DE LA SALUD. *Informática 2009.* [En línea] [Citado el: 2 de 4 de 2010.] http://informatica2009.sld.cu/Members/denis/gestion-de-la-arquitectura-del-proyecto-de-informatizacion-de-la-salud-1/at_download/trabajo.
54. **Kerl, Christian.** ckWebServicePlugin. *Symfony Project.* [En línea] [Citado el: 15 de 6 de 2009.] http://www.symfony-project.org/plugins/ckWebServicePlugin.
55. **José María Fuentes, Miguel Ángel Corella, Pablo Castells, Mariano Rico.** Generación semi-automática de servicios web. [En línea] [Citado el: 2 de 4 de 2010.] http://nets.ii.uam.es/~sws/publications/jsweb05-extended.pdf.
56. **Eichorn, Joshua.** phpDocumentor: The complete documentation solution for PHP. [En línea] [Citado el: 15 de 6 de 2009.] http://www.phpdoc.org/.
57. **PHP, Team.** SoapServer calls wrong function, although "SOAP action" header is correct. [En línea] http://bugs.php.net/49169.
58. **Morales, Annia Arencibia, Velázquez, Karel Gámez y González, Leonardo González.** *Centro de Control para el Sistema de Información para la Salud.* 2007.
59. *CRITERIO DE EXPERTOS: MÉTODO DELPHY.* **varios.**
60. **Valderrama, Rubén Peredo, y otros, y otros.** ARQUITECTURA PARA SISTEMAS DE EDUCACIÓN BASADA EN WEB USANDO PROGRAMACIÓN ORIENTADA A COMPONENTES. [En línea] [Citado el: 4 de 6 de 2009.] http://www2.pucpr.br/reol/index.php/DIALOGO?dd1=2033&dd99=pdf.

61. **Candillon, William.** phpaspect, Aspect-Oriented Programming for PHP. [En línea] 2009. http://code.google.com/p/phpaspect/.
62. **Blanco, Guilherme.** Transparent PHP AOP. *PHP Classes.* [En línea] http://www.phpclasses.org/browse/package/3215.html.
63. **Systems, Cisco.** AAA Overview. *Cisco IOS Security Configuration Guide, Release 12.2.* [En línea] [Citado el: 10 de 6 de 2009.] http://www.cisco.com/en/US/docs/ios/12_2/security/configuration/guide/scfaaa.html.
64. **Esperón, Maricela Torres.** Metodología para definir funciones profesionales. *Revista Cubana de Salud Pública.* 2008, Vol. 34.
65. El método Delphi. [En línea] [Citado el: 13 de 11 de 2009.] http://www.gtic.ssr.upm.es/encuestas/delphi.htm.

Anexos

Anexo 1: Ejemplo de uso del *sfAspectPlugin*

Para ilustrar el funcionamiento del *sfAspectPlugin* a continuación se pondrá un ejemplo de cómo puede ser utilizado. Supongamos que queremos guardar en los *log* de la aplicación, el nombre del método y los parámetros, de cualquier clase del modelo del proyecto que sea invocada.

Lo primero sería definir en el fichero *config_aspect.yml* la ubicación del directorio donde están las clases de nuestro modelo. El siguiente listado tiene la definición necesaria.

Listado 7	Ejemplo del fichero config_aspect.yml para aplicar los aspectos a las clases del modelo.

```
aspect:
  project_model:
    path:           %SF_LIB_DIR%/model
    recursive:      on
    exclude_dir:    []
    exclude_class:  []
    file_aspect:    [log]
```

Según la configuración establecida, el *plugin* haría lo siguiente:

- Solo se aplicarán los aspectos definidos en el fichero *log.xml* y no todos los definidos en el directorio *config/aspect*.
- Solo se le aplicarán los aspectos a las clases ubicadas en el directorio *lib/model* o en sus subdirectorios.
- Y no se excluirá ningún subdirectorio ni ninguna clase.

El siguiente paso sería crear el fichero *log.xml* con la definición de los aspectos. Para ganar en claridad se hará lo más sencillo posible. El siguiente listado muestra su contenido.

Listado 8	Contenido del fichero log.xml

```
<?xml version="1.0" encoding="utf-8"?>
<aspect>
    <pointcut auto="before">
    <![CDATA[
     $param = func_get_args(); //obtengo los parámetros que se le
                               //pasaron a la función.
  sfContext::getInstance()->getLogger()->
info(__CLASS__.'::'.__FUNCTION__.'('.var_export($param, true).')');
  //guardo en los log el nombre de la clase, la función invocada y los
  //parámetros que se le pasaron.
    ]]>
    </pointcut>
</aspect>
```

En el fichero se declara un solo *pointcut*, que define que se debe ejecutar este código antes de la ejecución de cualquier método. El *plugin* determina esto debido a la directiva *before* puesta en la declaración del *pointcut*.

Por último, solo queda implementar una pequeña aplicación para verificar el funcionamiento de lo realizado hasta aquí.

Se hará una clase llamada *PruebaAspect* con un solo método llamado prueba que devolverá una simple cadena de texto. Esta clase la se ubicará en el directorio *lib/model* del proyecto para que sea procesada por el *plugin*. El siguiente listado muestra su código fuente.

Listado 8 Clase PruebaAspect.

```
<?php
class PruebaAspect
{
    public function prueba($valor)
    {
       return 'Ha pasado el valor => '.$valor;
    }
}
```

A continuación se hará un módulo llamado prueba que solo tendrá la acción index. La definición de la clase es la siguiente:

Listado 8 Implementación de la clase pruebaActions.

```
class pruebaActions extends sfActions
{
    public function executeIndex()
    {
       $this->objeto = new PruebaAspect();//lo único que se hace es
                                          //pasar un objeto de tipo
                                          //PruebaAspect a la
                                          //plantilla.
    }
}
```

Y finalmente en la plantilla se pondría lo siguiente:

Listado 8 Contenido de la plantilla.

```
<h1>Prueba de sfAspectPlugin</h1>
<p>¡Este el resultado que se obtiene desde la clase del modelo:</p>
<p><?php echo $objeto->prueba('un valor cualquiera.')?>!</p>
```

Una última aclaración: en el código de ejemplo se invocan realmente dos métodos de la clase *PruebaAspect*: el constructor y el método "*prueba"*, pero solo se aplicará el aspecto a este último debido a que el constructor no ha sido definido en la clase y el *plugin* solamente aplica los aspectos a los métodos declarados. Si desea que también se tenga en cuenta el constructor, debe declararlo de forma explícita en la definición de la clase.

Después de esto si se observan los log de la aplicación, se verá una entrada similar a la siguiente:

Listado 8 Contenido en los log de la aplicación.

```
Nov 25 16:30:25 symfony [info ] PruebaAspect::prueba(array (0 => 'un valor cualquiera.')
```

Esto demuestra que se han aplicado los aspectos de forma correcta.

Luego puede ver en el directorio cache/sfAspectPlugin la nueva clase generada a partir de la clase *PruebaAspect* en un fichero con el nombre *PruebaAspect.aspect*. <un hash de la dirección>.php para que vea el nuevo código de la clase con los aspectos aplicados.

Listado 8 Contenido de la clase PruebaAspect con los aspectos aplicados.

```
<?php
class PruebaAspect
{
    public function prueba($valor)
    {
       /* AOP "before" Auto Code */
       $param = func_get_args(); //obtengo los parámetros que se le pasaron a la
                                 //función.
    sfContext::getInstance()->getLogger()->
           info(__CLASS__.'::'.__FUNCTION__.'('.var_export($param, true).')');
    //guardo en los log el nombre de la clase, la función invocada y
    //los parámetros que se le pasaron.

       return 'Ha pasado el valor => '.$valor;
    }
}
```

Anexo 2: Encuestas aplicadas al comité de expertos

2.1 Encuesta para determinar el nivel de competencia

Encuesta No.1 a Expertos
Competencias de Expertos

Nota:
El objetivo de esta encuesta es solamente investigativa. El responsable de esta encuesta se compromete a mantener total privacidad de la información recopilada.

Pregunta 1

Marque con una X, en una escala del 1 al 10 el valor que se corresponde con el grado de conocimiento o información que usted considera que tiene respecto a la **arquitectura y el modelo de desarrollo de las aplicaciones para el SISalud**. 1 indica que no tiene ningún conocimiento sobre el tema y 10 indica que tiene pleno conocimiento sobre él.

1	2	3	4	5	6	7	8	9	10

Pregunta 2

Señale con una X el nivel de influencia que ha tenido cada una de las fuentes indicadas en su conocimiento sobre la **arquitectura y el modelo de desarrollo de las aplicaciones para el SISalud**.

FUENTES DE ARGUMENTACION	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis teóricos realizados por usted			
Su experiencia obtenida			
Trabajos de autores nacionales			
Trabajos de autores extranjeros			
Su propio conocimiento del estado del problema en el extranjero			
Su intuición			

2.2 Encuesta para determinar los principales problemas

Encuesta No.2 a Expertos

Nota:

El objetivo de esta encuesta es solamente investigativa. El responsable de esta encuesta se compromete a mantener total privacidad de la información recopilada.

Pregunta 1

¿Cuáles considera usted que son los principales problemas -dentro de la arquitectura- que afectan a todas las aplicaciones que se desarrollan para el SISalud?

Pregunta 2

Evalúe cada uno de los aspectos anteriores de acuerdo al nivel de importancia que usted le confiere. (1 a 10), donde:

(1) Poca importancia.

(10) Mucha importancia.

Actividades	1	2	3	4	5	6	7
Nivel de importancia							

2.3 Encuesta para la evaluación de la solución propuesta

Encuesta No. 3 a Expertos

Nombre:

Nota:

El objetivo de esta encuesta es solamente investigativa. El responsable de esta encuesta se compromete a mantener total privacidad de la información recopilada.

Pregunta 1

Evalúe los elementos implementados en el slSISaludPlugin según los siguientes aspectos.

a. Los mecanismos implementados para que una aplicación cliente consuma los servicios de otro componente del SISalud (**Integración con otros componentes**) son:
   ___ Insuficientes.
   ___ Necesarias pero no suficientes para el desarrollo de un componente.
   ___ Necesarias y suficientes para el desarrollo de un componente
   Explique brevemente su opinión si lo considera necesario:
b. Los mecanismos implementados para que una aplicación brinde servicios a otro componente del SISalud (**Implementación de la interfaz de servicio de los componentes**) son:
   ___ Insuficientes.
   ___ Necesarias pero no suficientes para el desarrollo de un componente.
   ___ Necesarias y suficientes para el desarrollo de un componente
   Explique brevemente su opinión si lo considera necesario:
c. Los mecanismos implementados para la integración con el SAAA y gestión de la seguridad (**Integración con el SAAA**) son:
   ___ Insuficientes.
   ___ Necesarias pero no suficientes para el desarrollo de un componente.
   ___ Necesarias y suficientes para el desarrollo de un componente
   Explique brevemente su opinión si lo considera necesario:
d. Los mecanismos implementados para la gestión de eventos dentro de un componente del SISalud (**Gestión de eventos en el SISalud**) son:
   ___ Insuficientes.
   ___ Necesarias pero no suficientes para el desarrollo de un componente.
   ___ Necesarias y suficientes para el desarrollo de un componente
   Explique brevemente su opinión si lo considera necesario:
e. El modelo de excepciones propuesto (**Modelo de excepciones**) es:
   ___ Insuficientes.
   ___ Necesarias pero no suficientes para el desarrollo de un componente.

___ Necesarias y suficientes para el desarrollo de un componente

Explique brevemente su opinión si lo considera necesario:

Pregunta 2

¿Considera usted que la utilización de la Programación Orientada a Aspecto para la realización de la auditoría en los sistemas es útil?

___ Si ___ No

¿Por qué?

Pregunta 3

¿Cree usted que brindar la posibilidad de que se lancen eventos desde de una aplicación es útil?

___ Si ___ No

¿Por qué?

Pregunta 4

Evalúe los siguientes elementos según el nivel de importancia que usted le confiera, donde:

(1) significa poco valor.

(10) significa el máximo valor posible.

Criterio	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Integración con otros componentes											
Implementación de la interfaz de servicio de los componentes											
Integración con el SAAA											
Gestión de eventos en el SISalud											
Modelo de excepciones											