Universidad de las Ciencias Informáticas

Facultad 1

Título: Integración de los sistemas de la Plataforma de Migración a Software Libre y Código Abierto

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor(es): Abel García Vitier, Jailen García González

Tutor: Ing. Yoandy Pérez Villazón

Co-tutora: Ing. Lisandra Cala Hernández

Ciudad de La Habana, Cuba, Junio 2011.

"Año 52 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos al departamento SIMAYS de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Abel García Vitier

_____

Jailen García González

_____

Ing. Yoandy Pérez Villazón

_____

Ing. Lisandra Cala Hernández

_____

RESUMEN

Los procesos de migración que lleva a cabo el departamento SIMAYS en diferentes instituciones, requieren de un sistema centralizado que permita su gestión automatizada. Para ello, se necesita vincular las distintas aplicaciones que se usan en estos procesos, permitiendo su funcionamiento como un todo. Esta investigación tiene como objetivo integrar los sistemas componentes de la Plataforma de Migración a Software Libre y Código Abierto (PMSWL). Una vez realizado el estudio de los mecanismos de integración, los estilos arquitectónicos y las tecnologías más apropiadas, se define una arquitectura que combina los estilos SOA y Presentación Desacoplada para desarrollar los componentes de esta integración. Con el uso de la metodología de desarrollo ágil SXP, se construyeron los sistemas de control de seguridad y de manejo de interfaces. El primero se encarga de proveer las identidades y el control de acceso de manera centralizada, mientras que la función del segundo es gestionar las interfaces correspondientes a cada sistema de la PMSWL, usando para cada uno un módulo de interfaz de usuario.

PALABRAS CLAVE

Código abierto, control de acceso, integración de sistemas, migración, plataforma de migración, proveedor de identidad, software libre.

Índice

INTRODUCCIÓN

A medida que las empresas van desarrollando e implantando sistemas de gestión normalizados, se evidencia más la necesidad de racionalizar los esfuerzos, recursos y costos destinados a todos ellos. El desarrollo, inicialmente paralelo, de distintas aplicaciones como sistemas de gestión independientes les ha restado eficiencia, al provocar problemas de duplicidad documental, solapamiento de costos y actividades. La carga de trabajo que implica la gestión independiente de los distintos sistemas ha presionado a las empresas en el mundo a desarrollar esquemas que permitan reducirla a través de la integración y la simplificación.

La integración de sistemas tiene una serie de ventajas respecto a una mejor organización del trabajo, a la optimización de los recursos y al ahorro en costos, como son la sinergia entre los sistemas, la simplificación de la documentación y su gestión, la optimización de la formación del personal gracias a la integración de procesos, la mejora de la percepción y de la implicación del personal en los sistemas de gestión. De igual forma favorece el incremento de la capacidad de reacción de la organización frente a las nuevas necesidades o expectativas de las partes interesadas y la comunicación tanto interna como externa.

A pesar de sus ventajas, la integración de sistemas no está libre de riesgos. Las dificultades para la unificación pueden residir en aspectos muy distintos como las resistencias normales a los procesos de cambio, que exigen un esfuerzo organizativo y humano importante; los distintos grados de implantación de los propios sistemas; el grado de compatibilidad entre los principios que guían cada sistema; la necesidad de recursos y capacidades adicionales para planificar y ejecutar el plan de integración; las propias estrategia, estructura y cultura de la empresa y la dificultad para elegir el nivel de integración adecuado al nivel de madurez de la organización.

La migración a software libre y código abierto constituye más que un cambio tecnológico, un cambio en la manera de pensar; el éxito de la misma está basado en la efectividad, eficiencia e integridad del propio proceso. En Cuba la migración constituye una necesidad para alcanzar la soberanía tecnológica, este proceso es dirigido por el Grupo Ejecutivo que coordina las acciones del Grupo de Capacitación, el

Grupo Legal, el Grupo de Divulgación y el Grupo Técnico Nacional, este último asumido por el Departamento de Sistemas Operativos y Desarrollo de Tecnologías Libres (SODTL) de la Universidad de las Ciencias Informáticas (UCI) y que se materializa en el departamento Servicios Integrales de Migración, Asesoría y Soporte (SIMAYS).(Cala Hernández 2010)

Existe una metodología para la migración que está soportada sobre flujos de trabajo que abarcan un conjunto limitado de tareas, que conducen al objetivo final del proceso en las entidades involucradas. Sin embargo, resulta inoperante e ineficiente la ejecución de la mayoría de las tareas con el uso de sistemas que actúan por separado. Actualmente se pierde tiempo en el análisis de la información recopilada del levantamiento en las empresas; en la búsqueda de herramientas alternativas para la migración, cuando por lo general son las mismas; en la realización de informes para enviar al grupo ejecutivo informando el avance de la migración (los informes en algunos casos van con información irreal pues no se tienen los indicadores para medir la migración) y en las búsquedas en internet para validar el hardware detectado. También se torna difícil planificar el proceso en las empresas de manera ordenada y poder medir los avances del mismo bajo el uso de indicadores reales.

Los procesos de migración que lleva a cabo el departamento SIMAYS en las distintas instituciones, no cuentan con un sistema centralizado que permita su gestión automatizada. Se necesita integrar las distintas aplicaciones que se usan en estos procesos y fueron propuestas a partir de una investigación para que funcionasen como una plataforma. Luego el **problema científico** es ¿cómo integrar los sistemas componentes de la plataforma de migración a software libre y código abierto?

Para dar solución al mismo se define como **objetivo general** integrar los sistemas componentes de la plataforma de migración a software libre y código abierto. El **objeto de estudio** de la investigación es el conjunto de mecanismos para la integración de sistemas y el **campo de acción**, el conjunto de mecanismos para la integración de sistemas mediante servicios web. Se pretende cumplir el objetivo general a partir los siguientes **objetivos específicos**:

1. Caracterizar los mecanismos actuales para la integración de sistemas mediante servicios web.
2. Describir y analizar la plataforma de migración a software libre y código abierto (PMSWL)
3. Desarrollar los componentes para la integración de los sistemas de la PMSWL.

4. Diseñar y ejecutar pruebas a la PMSWL para asegurar la calidad del producto.

En esta investigación la **idea a defender** es que la integración de los sistemas componentes de la plataforma de migración a software libre y código abierto simplificará y hará más productivo y eficiente un proceso de migración basado en la metodología cubana.

Los anteriores objetivos se concretan en las siguientes **tareas de la investigación:**

1. Examen del estado del arte acerca de la integración de sistemas.
2. Análisis de los mecanismos de integración, identificando aquellos elementos que puedan contribuir a la solución del problema.
3. Definición de los requisitos de la plataforma de migración a software libre y código abierto orientados a la integración de sus sistemas.
4. Definición de una arquitectura para la plataforma.
5. Análisis y diseño de los componentes para la integración.
6. Implementación robusta y segura de los componentes para la integración.
7. Pruebas a los componentes desarrollados para la integración.
8. Pruebas al funcionamiento global de la plataforma.

Métodos Teóricos:

Analítico sintético: El análisis es una operación intelectual que posibilita descomponer mentalmente un todo complejo en sus partes y cualidades. El análisis permite la división mental del todo en sus múltiples relaciones y componentes. La síntesis es la operación inversa, que establece mentalmente la unión entre las partes, previamente analizadas y posibilita descubrir relaciones y características generales entre los elementos de la realidad. Se utiliza en la investigación de los mecanismos de integración de sistemas ya existentes para comprender su funcionamiento y poder identificar qué elementos pueden ser útiles para dar solución al problema.

Inductivo deductivo: La inducción se puede definir como una forma de razonamiento por medio de la cual se pasa del conocimiento de casos particulares a un conocimiento más general que refleja lo que

hay de común en los fenómenos individuales y la deducción es una forma de razonamiento, mediante el cual se pasa de un conocimiento general a otro de menor nivel de generalidad. Se aplica durante la especificación de la arquitectura y el uso de patrones de diseño para resolver problemas particulares del sistema en desarrollo.

Este documento está estructurado en 5 capítulos, a continuación se brinda una breve descripción de lo que trata cada uno:

Capítulo I: *Elementos clave en la definición de un mecanismo de integración*: Se realiza un análisis del tema a tratar, se describen estilos usados en el diseño de una arquitectura de integración, así como las tecnologías necesarias para implementar el diseño.

Capítulo II: *Descripción y análisis de la PMSWL*: Se realiza la concepción de la plataforma, el modelo de negocio, la definición de los requisitos para la integración y se describe la arquitectura de la misma.

Capítulo III: *Desarrollo ágil del Sistema de Control de Seguridad (SCS)*: Se realiza el diseño e implementación de un sistema para proveer las identidades y el control de acceso de manera centralizada.

Capítulo IV: *Desarrollo ágil del Sistema para el Manejo de Interfaces (SMI)*: Se realiza el diseño e implementación de un sistema para gestionar las interfaces correspondientes a cada sistema de la PMSWL.

Capítulo V: *Discusión de Resultados*: Se plasman los casos de prueba de aceptación a las que fueron sometidos los componentes desarrollados. Se exponen los resultados obtenidos y las pruebas de integración.

CAPÍTULO 1: ELEMENTOS CLAVE EN LA DEFINICIÓN DE UN MECANISMO DE INTEGRACIÓN.

En la medida en que una organización define e implanta un sistema de gestión certificable se hace más evidente la necesidad de racionalizar los esfuerzos, costos y recursos destinados al mismo. Sobre todo cuando las normas de referencia en las que se basa, comparten requisitos en un porcentaje importante, y la metodología de gestión es 100% idéntica. Por lo tanto, el planteamiento de optimizar recursos, costos y esfuerzos vendrá por la integración común de todos aquellos conceptos cuya gestión tiene aspectos y requisitos comunes. El objetivo no es otro que evitar duplicidades, optimizar recursos y simplificar al máximo la gestión de todos los sistemas.

Integración es el proceso a través del cual la organización aprende a introducir criterios y especificaciones en sus procesos y en sus sistemas, de modo que satisfagan a todos sus clientes (internos, externos, institucionales, partes interesadas, etc.) de forma simultánea, ahorrando costos y esfuerzos, con un espíritu innovador, autocrático y comprometido con la mejora continua. (Vargas 2008)

Cualquier estrategia de implantación que tenga como objetivo la integración de los sistemas deberá tener en cuenta su relación con los procesos claves y relevantes que cruzan horizontalmente y verticalmente toda organización. Los sistemas deben estar supeditados a los procesos de gestión relacionados y servir de herramientas estructuradas para la gestión de los mismos.

La integración de sistemas de información representa un desafío continuo para las organizaciones. Las innovaciones en la tecnología avanzan rápidamente, rebasando el tiempo de vida útil de las aplicaciones que se adquieren. Como consecuencia de estas innovaciones, los ambientes se vuelven heterogéneos y se comienzan a disponer de diferentes herramientas informáticas que responden a necesidades concretas en un tiempo determinado.

Los objetivos de una estrategia de integración de sistemas han de incluir:

- Reducir el número de procesos necesarios para cumplir con los objetivos de negocio.

- Proporcionar al sistema de información la flexibilidad necesaria para adaptarse al cambio.
- Dotar a la plataforma de la capacidad para evolucionar a un costo razonable.

Para integrar sistemas existentes en una plataforma es necesario definir una solución para los requisitos técnicos y operacionales de la misma, a partir del diseño de una arquitectura. En este proceso se establecen qué componentes forman parte del sistema, cómo se relacionan entre ellos y cómo mediante su interacción se lleva a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados como seguridad, disponibilidad, eficiencia o usabilidad.

Para diseñar la arquitectura de un sistema es importante tener en cuenta los intereses de los distintos agentes que participan. Estos agentes son los usuarios del sistema, el propio sistema y los objetivos del negocio. Cada uno de ellos impone requisitos y restricciones que deben ser tenidos en cuenta en el diseño de la arquitectura que pueden llegar a entrar en conflicto, por lo que se debe alcanzar un compromiso entre los intereses de cada participante.

El objetivo final de una arquitectura es identificar los requisitos que producen impacto en la estructura del sistema y reducir los riesgos asociados con la construcción del mismo. La arquitectura debe soportar los cambios futuros del software, del hardware y de la funcionalidad demandada por los clientes. En síntesis una arquitectura debería:

- Mostrar la estructura del sistema pero ocultar sus detalles.
- Realizar todas las funcionalidades.
- Satisfacer en la medida de lo posible los intereses de los agentes.
- Ocuparse de los requisitos funcionales y de calidad.
- Determinar el tipo de sistema a desarrollar.
- Determinar los estilos arquitectónicos que se usarán.
- Tratar las principales cuestiones transversales.

La selección de un tipo de aplicación determina en cierta medida el estilo arquitectónico que se va a usar. El estilo arquitectónico es en esencia la partición más básica del sistema en bloques y la

forma en que se relacionan estos bloques. Existen disímiles estilos arquitectónicos que serán caracterizados en este capítulo para poder seleccionar cuál o cuáles pueden usarse en el diseño de la arquitectura de la PMSWL.

1.1 Estilos arquitectónicos.

Cuando el arquitecto le da forma a la arquitectura de una aplicación, el estilo arquitectónico es su herramienta básica. Un estilo arquitectónico puede entenderse como un conjunto de principios que definen a alto nivel un aspecto de la aplicación. Este viene definido por un conjunto de componentes, conexiones entre dichos componentes y un conjunto de restricciones sobre cómo se comunican dos componentes cualesquiera una vez conectados. Los estilos arquitectónicos pueden organizarse en torno al aspecto de la aplicación sobre el que se centran. Los principales aspectos son: comunicaciones, despliegue, dominio, interacción y estructura.

Generalmente lo que sucede es que una arquitectura combina varios estilos arquitectónicos para obtener las ventajas de cada uno. Los estilos arquitectónicos son indicaciones abstractas que son instanciadas en una aplicación real y esta es la principal diferencia entre estilo arquitectónico y patrón de diseño. Los patrones de diseño son descripciones estructurales y funcionales de cómo resolver un problema concreto mediante orientación a objetos, mientras que los estilos arquitectónicos no están atados a ningún paradigma de programación específico.

A continuación se presentan los principales estilos arquitectónicos estudiados, a partir de sus características, beneficios y factores que pueden determinar su uso:

1.1.1 Cliente – Servidor.

El estilo cliente/servidor define una relación entre dos aplicaciones en las cuales una de ellas (cliente) envía peticiones a la otra (servidor fuente de datos). (Meier J.D. et al. 2009)

Características:

- Es un estilo para sistemas distribuidos.

- Divide el sistema en una aplicación cliente, una servidora y una red que las conecta.
- Describe una relación entre el cliente y el servidor en el cual el primero realiza peticiones y el segundo envía respuestas.
- Puede usar amplio rango de protocolos y formatos de datos para comunicar la información.

Principios Clave:

- El cliente realiza una o más peticiones, espera por las respuestas y las procesa a su llegada.
- El cliente se conecta normalmente a solo uno o un número reducido de servidores al mismo tiempo.
- El cliente interactúa directamente con el usuario, por ejemplo mediante una interfaz gráfica.
- El servidor no realiza ninguna petición al cliente.
- El servidor envía los datos en respuesta a las peticiones realizadas por los clientes conectados.
- El servidor normalmente autentifica y verifica primero al usuario y después procesa la información y envía los resultados.

Beneficios:

- Más seguridad pues los datos se almacenan en el servidor que generalmente ofrece mayor control sobre la misma.
- Acceso centralizado a los datos almacenados en el servidor, lo que facilita su actualización.
- Facilidad de mantenimiento, pues los roles y las responsabilidades se distribuyen entre los distintos servidores a través de la red, lo que permite que un cliente no se vea afectado por un error en un servidor específico.

Cuándo usarlo:

- La aplicación se basa en un servidor y soportará múltiples clientes.
- La aplicación está normalmente a un uso local y red controlada.
- Se implementan procesos de negocio que se usarán a lo largo de toda la organización.
- Se quiere centralizar el almacenamiento, las salvas y el mantenimiento de la aplicación.
- La aplicación debe soportar distintos tipos de clientes y dispositivos.

1.1.2 Basado en Componentes.

El estilo de arquitectura basado en componentes se acerca al diseño de un sistema como un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver el problema. (Meier J.D., Hill David, Alex Hommer, and Jason Taylor 2009)

Características:

- Es un estilo para diseñar aplicaciones a partir de componentes individuales.
- Se centra en la descomposición del sistema en componentes con interfaces muy bien definidas.
- Define una aproximación al diseño a través de componentes que se comunican mediante interfaces que exponen métodos, eventos y propiedades.

Principios clave:

- Los componentes son diseñados de manera que puedan reutilizarse en distintos escenarios en distintas aplicaciones aunque algunos componentes son diseñados para una tarea específica.
- Los componentes son diseñados para funcionar en diferentes entornos y contextos. Toda la información debe ser pasada al componente, en lugar de incluirla en él o que este acceda a ella.
- Los componentes pueden ser extendidos a partir de otros componentes para ofrecer nuevos comportamientos.
- Los componentes poseen interfaces que permiten al código usar su funcionalidad y no revelan detalles internos de los procesos que realizan o de su estado.
- Los componentes están diseñados para ser lo más independiente posible de otros componentes, por lo que pueden ser desplegados sin afectar a otros componentes o sistemas.

Beneficios:

- Fácil despliegue, debido a la posibilidad de sustitución de un componente por su nueva versión sin afectar a otro componente o al sistema.
- Reducción de costos, pues pueden usarse componentes de terceros para disminuir los costos de desarrollo y mantenimiento.

- Reusabilidad de los componentes, dada su independencia del contexto y la posibilidad de utilización en otros sistemas o aplicaciones.
- Reducción de la complejidad, gracias al uso de contenedores de componentes que realizan la gestión del ciclo de vida.

Cuándo usarlo:

- Se tienen componentes que le sirvan al sistema o pueden conseguirse.
- La aplicación ejecutará generalmente procedimientos con pocos o ningún dato de entrada.
- Se quiere poder combinar componentes escritos en diferentes lenguajes.
- Se quiere crear una arquitectura que permita el reemplazo o actualización de uno de sus componentes de forma sencilla.

1.1.3 En Capas (N-Layer).

Este estilo arquitectónico se basa en una distribución jerárquica de los roles y responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de interacción con otras capas, mientras que las responsabilidades indican la funcionalidad que implementan. (Hill 2007)

Características:

- Descomposición de los servicios, de manera que la mayoría de las interacciones se realizan sólo entre capas vecinas.
- Las capas de una aplicación pueden residir en la misma computadora o pueden estar distribuidas entre varios equipos.
- Los componentes de cada capa se comunican con los componentes de otras capas, a través de interfaces bien conocidas.
- Cada nivel agrega las responsabilidades y abstracciones del nivel inferior.
- Muestra una vista completa del modelo y al mismo tiempo brinda suficientes detalles para entender las relaciones entre capas.
- No supone sobre los tipos de datos, métodos, propiedades y sus implementaciones.

- Separa claramente la funcionalidad de cada capa.

Principios Clave:

- Cada capa contiene la funcionalidad relacionada sólo con las tareas de esa capa.
- Las capas inferiores no tienen dependencias de las superiores.
- La comunicación entre capas está basada en una abstracción que proporciona bajo acoplamiento entre capas.

Beneficios:

- Abstracción, pues los cambios se realizan a alto nivel y puede incrementarse o reducirse el nivel de abstracción usado en cada capa del modelo.
- Aislamiento, pues se pueden realizar actualizaciones en el interior de las capas sin que esto afecte al resto del sistema.
- Rendimiento, debido a la distribución de las capas en diferentes niveles físicos, mejorando así la escalabilidad y la tolerancia a fallos.
- Capacidad que tiene cada capa de que puedan realizarse pruebas sobre su interfaz bien definida y la habilidad de cambiar entre diferentes implementaciones de una capa.
- Independencia al eliminar la necesidad de considerar el hardware y el despliegue, así como dependencias con sus interfaces externas.

Cuándo usarlo:

- Ya se tienen construidas capas de una aplicación anterior, y estas pueden reutilizarse o integrarse.
- Existen aplicaciones que exponen su lógica de negocio mediante interfaces de servicios.
- La aplicación es compleja y el alto nivel de diseño requiere la separación, para que los distintos equipos puedan concentrarse en distintas áreas de funcionalidad.
- La aplicación debe soportar distintos tipos de clientes y dispositivos.
- Se quieren implementar reglas y procesos de negocio complejos o configurables.

1.1.4 Presentación Desacoplada.

El estilo de presentación separada indica cómo debe realizarse el manejo de las acciones del usuario, la manipulación de la interfaz y los datos de la aplicación. Este estilo separa los componentes de la interfaz del flujo de datos y de la manipulación. (Fowler 2006)

Características:

- Es un estilo para diseñar aplicaciones, basado en patrones de diseño conocidos.
- Separa la lógica para el manejo de la interacción de la representación de los datos con que trabaja el usuario.
- Permite a los diseñadores crear una interfaz gráfica, mientras los desarrolladores escriben el código para su funcionamiento.
- Ofrece un mejor soporte para las pruebas, pues se pueden probar los comportamientos individuales.

Principios Clave:

- Separa el procesamiento de la interfaz en distintos roles.
- Permite construir *mocks* (objetos simulados) que replican el comportamiento de otros objetos durante las pruebas.
- Utiliza eventos para notificar a la vista que hay datos del modelo que han sido modificados.
- El controlador maneja los eventos disparados desde los controles de usuario en la vista.

Beneficios:

- Capacidad de prueba, pues en las implementaciones comunes los roles son simplemente clases que pueden ser probadas y reemplazadas por *mocks* que simulen su comportamiento.
- Reusabilidad, de manera que los controladores pueden ser aprovechados en otras vistas compatibles y viceversa.

Cuándo usarlo:

- Se quieren mejorar las pruebas y el mantenimiento de la funcionalidad de la interfaz.

- Se quiere separar la tarea de crear la interfaz de la lógica que la maneja.
- No se quiere que la interfaz contenga código de procesamiento de eventos.
- El código de procesamiento de la interfaz no implementa lógica de negocio.

1.1.5 N-Niveles (N-Tier).

Este estilo arquitectónico define la separación de la funcionalidad en segmentos o niveles físicos separados, es similar al estilo N-Capas, sólo que sitúa cada segmento en una máquina diferente. En este caso se habla de niveles físicos. (Dahan 2007)

Características:

- Separación de niveles físicos (servidores normalmente) por razones de seguridad, escalabilidad o simplemente necesidad.

Principios Clave:

- Es un estilo para definir el despliegue de las capas de la aplicación.
- Descomposición funcional de las aplicaciones, componentes de servicio y despliegue distribuido, que ofrece mejor escalabilidad, disponibilidad, rendimiento, manejabilidad y uso de recursos.
- Cada nivel es completamente independiente de los otros niveles excepto del inmediatamente inferior.
- Tiene al menos tres niveles lógicos separados. Cada capa implementa funcionalidades específicas y está físicamente separada en distintos servidores.
- Una capa es desplegada en un nivel si uno o más servicios o aplicaciones dependen de la funcionalidad expuesta por dicha capa.

Beneficios:

- Mantenimiento, porque cada nivel es independiente de los otros y las actualizaciones y cambios pueden ser llevados a cabo sin afectar la aplicación como un todo.
- Escalabilidad, porque los niveles están basados en el despliegue de capas y realizar el escalado de la aplicación es bastante directo.

- Disponibilidad, dado que las aplicaciones pueden redundar cualquiera de los niveles y ofrecer así tolerancia ante fallos.

Cuándo usarlo:

- Difieren los requisitos de procesamiento de las capas de la aplicación.
- Difieren los requisitos de seguridad de las capas de la aplicación.
- Se quiere compartir la lógica del negocio entre varias aplicaciones.
- Se posee suficiente hardware para desplegar el número necesario de servidores en cada nivel.

1.1.6 Arquitectura Orientada al Dominio (DDD)

DDD (Domain Driver Design) además de ser un estilo arquitectónico, es también una forma de afrontar los proyectos a nivel de trabajo del equipo de desarrollo, ciclo de vida del proyecto, identificación del lenguaje ubicuo a utilizar con los expertos del negocio, etc. Sin embargo, DDD también identifica una serie de patrones de diseño y estilo de arquitectura concreto.

DDD es una aproximación concreta para diseñar software basándose en la importancia del dominio del negocio, sus elementos y comportamientos y las relaciones entre ellos. Es muy indicado para diseñar aplicaciones empresariales complejas donde es esencial la definición de un Modelo de Dominio expresado en el lenguaje propio de los expertos del dominio del negocio real (lenguaje ubicuo). (Laribee 2009)

Características:

- Identifica un grupo de patrones de arquitectura importantes a tener en cuenta en el diseño de una aplicación, como son:
  - Repository
  - Entity
  - Aggregate
  - Value-Object
  - Unit Of Work

    - Services
- En DDD también es fundamental el desacoplamiento entre componentes.

Principios Clave:

- Debe tenerse buen entendimiento del dominio de negocio que se quiere modelar, o conseguir el mismo adquiriéndolo de los expertos del dominio real.
- Todo el equipo de desarrollo debe tener contacto con los expertos del dominio (expertos funcionales) para modelar correctamente el mismo.
- Los arquitectos, desarrolladores, líder de proyecto y probadores deben acordar hacer uso de un único lenguaje sobre el dominio del negocio, centrado en cómo los expertos de dicho dominio articulan su trabajo.
- El corazón del software es el Modelo de Dominio que permite al equipo de desarrollo encontrar de manera rápida áreas incorrectas en el software a analizar.
- La creación del lenguaje ubicuo no es sólo un ejercicio de aceptar información de los expertos.
- Aunque proporciona muchos beneficios técnicos, este sólo debe aplicarse en dominios complejos donde el modelo y los procesos lingüísticos proporcionan claros beneficios en la comunicación de la información compleja y en la formulación de un entendimiento común.
- La complejidad arquitectónica es mayor que una aplicación orientada a datos aunque ofrece mayor mantenimiento y desacoplamiento entre componentes.

Beneficios:

- Comunicación, debido a que todas las partes de un equipo de desarrollo pueden usar el modelo de dominio y las entidades que define para comunicar conocimiento del negocio y los requerimientos, haciendo uso de un lenguaje común.
- Extensibilidad, dada porque la capa del dominio es el corazón del software y por tanto, estará completamente desacoplada de las capas de infraestructura, siendo más fácil de extender o evolucionar la tecnología del software.
- Facilita las pruebas y el mocking, debido a la tendencia en el diseño de desacoplar los objetos de las diferentes capas de la arquitectura.

Cuándo usarlo:

- Considerar DDD en aplicaciones complejas con mucha lógica del negocio, dominios complejos y donde se desea mejorar la comunicación y minimizar los malentendidos en la comunicación del equipo de desarrollo.
- Es una aproximación ideal en escenarios empresariales grandes y complejos que son difíciles de manejar con otras técnicas.

1.1.7 Orientado a Objetos.

Es un estilo que define al sistema como un conjunto de objetos que cooperan entre sí en un lugar. Los objetos son discretos, independientes y poco acoplados, se comunican mediante interfaces y permiten enviar y recibir mensajes. (Meier J.D., Hill David, Alex Hommer, and Jason Taylor 2009)

Características:

- Es un estilo para diseñar aplicaciones, basado en un número de unidades lógicas y código reutilizable.
- Describe el uso de objetos que contienen datos y el comportamiento para trabajar con esos datos, además tienen un rol distinto.
- Hace hincapié en la reutilización a través de la encapsulación, la modularidad, el polimorfismo y la herencia.
- Contrasta con el acercamiento procedimental donde hay una secuencia predefinida de tareas y acciones. El enfoque orientado a objetos emplea el concepto de objetos interactuando unos con otros para realizar las tareas.

Principios Clave:

- Permite reducir una operación compleja mediante generalizaciones que mantienen las características de la operación.
- Los objetos se componen de otros objetos y eligen esconder estos objetos internos de otras clases o exponerlos como simples interfaces.
- Los objetos heredan de otros y usan la funcionalidad de sus objetos base o redefinen dicha funcionalidad para implementar un nuevo comportamiento, la herencia facilita el

mantenimiento y la actualización, pues los cambios se propagan desde el objeto base a sus herederos automáticamente.

- Los objetos exponen la funcionalidad sólo a través de métodos, propiedades y eventos, ocultando detalles internos como el estado o las referencias a otros objetos. Esto facilita la actualización y reemplazo de objetos asumiendo que sus interfaces son compatibles sin afectar al resto.
- Los objetos son polimórficos, o sea, en cualquier punto donde se espere un objeto base se puede poner un objeto heredero.
- Los objetos se desacoplan de los objetos que los usan, implementando una interfaz que estos últimos conocen. Esto permite proporcionar otras implementaciones sin afectar a los objetos consumidores de la interfaz.

Beneficios:

- Comprensión, porque el diseño orientado a objetos define una serie de componentes mucho más cercanos a los objetos en el mundo real.
- Reusabilidad, pues el polimorfismo y la abstracción permiten definir contratos en interfaces y cambiar las implementaciones de forma transparente.
- Capacidad de prueba dada la encapsulación de los objetos.
- Extensibilidad debido a la encapsulación, el polimorfismo y la abstracción.

Cuándo usarlo:

- Se quiere modelar la aplicación basándose en objetos reales y sus acciones.
- Existen objetos que encajan en el diseño y con los requisitos operacionales.
- Se necesita encapsular la lógica y los datos juntos, de forma transparente.

1.1.8 Orientación a Servicios (SOA)

El estilo orientado a servicios permite a una aplicación ofrecer su funcionalidad como un conjunto de servicios para que sean consumidos. Los servicios usan interfaces estándares que pueden ser invocadas,

publicadas y descubiertas. Se centra en proporcionar un esquema basado en mensajes con operaciones de nivel de aplicación y no de componente o de objeto. (Endrei Mark et al.)

Características:

- La interacción con el servicio está muy desacoplada.
- Puede empaquetar procesos de negocio como servicios.
- Los clientes y otros servicios pueden acceder a servicios locales corriendo en el mismo nivel.
- Los clientes y otros servicios acceden a los servicios remotos a través de la red.
- Pueden usar un amplio rango de protocolos y formatos de datos.

Principios Clave:

- Los servicios son autónomos, o sea, cada servicio se mantiene, se desarrolla, se despliega y se versiona independientemente.
- Los servicios pueden estar situados en cualquier nodo de una red local o remota mientras esta soporte los protocolos de comunicación necesarios.
- Cada servicio es independiente del resto de servicios y puede ser reemplazado o actualizado sin afectar a las aplicaciones que lo usan mientras la interfaz sea compatible.
- Los servicios comparten esquemas y contratos para comunicarse en vez de clases.
- La compatibilidad se basa en políticas que definen funcionalidades como el mecanismo de transporte, el protocolo y la seguridad.

Beneficios:

- Alineamiento con el dominio, pues la reutilización de servicios comunes con interfaz estándar aumenta las oportunidades tecnológicas y de negocio.
- Abstracción, debido a que los servicios son autónomos y se accede a ellos a través de un contrato formal que proporciona bajo acoplamiento y abstracción.
- Descubrimiento, pues los servicios exponen descripciones que permiten a otras aplicaciones y servicios encontrarlos y determinar su interfaz automáticamente.

Cuándo usarlo:

- Se tiene acceso a servicios adecuados o pueden comprarse.
- Se quiere construir aplicaciones que compongan múltiples servicios en una interfaz única.
- Se necesita soportar comunicación basada en mensajes para segmentos de la aplicación.
- Se necesita exponer funcionalidad de forma independiente de la plataforma.
- Se necesitan utilizar servicios federados como servicios de autenticación.
- Se quieren exponer servicios que puedan ser descubiertos y usados por clientes que no tuvieran conocimiento previo de sus interfaces.
- Se quieren soportar escenarios de interoperabilidad e integración.

1.1.9 Bus de Servicios (Mensajes)

Este estilo define un sistema de software que puede enviar y recibir mensajes usando uno o más canales, de forma que las aplicaciones puedan interactuar sin conocer detalles específicos la una de la otra. (Trowbridge et al. 2004)

Características:

- Es un estilo para diseñar aplicaciones donde la interacción entre las mismas se realiza a través del paso de mensajes por un canal de comunicación común.
- Las comunicaciones entre aplicaciones suelen ser asíncronas.
- Se implementa a menudo usando un sistema de mensajes como MSMQ.
- Muchas implementaciones consisten en aplicaciones individuales que se comunican usando esquemas comunes y una infraestructura compartida para el envío y recepción de mensajes.

Principios Clave:

- Toda la comunicación entre aplicaciones se basa en mensajes que usan esquemas comunes.
- Las operaciones complejas pueden ser creadas combinando un conjunto de operaciones más simples que realizan determinadas tareas.
- Como la interacción con el bus está basado en esquemas comunes y mensajes, se pueden añadir o eliminar aplicaciones del bus para cambiar la lógica usada para procesar los mensajes.

- Al usar un modelo de comunicación con mensajes basados en estándares, se puede interactuar con aplicaciones desarrolladas para distintas plataformas.

Beneficios:

- Expansión, debido a que las aplicaciones pueden ser añadidas o eliminadas del bus sin afectar al resto de las aplicaciones existentes.
- Reducción de la complejidad, dado que cada aplicación sólo necesita conocer cómo comunicarse con el bus.
- Mejor rendimiento, pues no hay intermediarios en la comunicación entre dos aplicaciones, sólo la limitación de lo rápido que entregue el bus a. los mensajes.
- Escalabilidad, debido a que muchas instancias de la misma aplicación pueden ser asociadas al bus para dar servicio a varias peticiones al mismo tiempo.
- Simplicidad, pues cada aplicación sólo tiene que soportar una conexión con el bus, en lugar de varias conexiones con el resto de las aplicaciones.

Cuándo usarlo:

- Se tienen aplicaciones que interactúan unas con otras para realizar tareas.
- Se implementa una tarea que requiere interacción con aplicaciones externas.
- Se implementa una tarea que requiere interacción con otras aplicaciones desplegadas en entornos diferentes.
- Se tienen aplicaciones que realizan tareas separadas y se quieren combinar las mismas en una sola operación.

1.2 Tecnologías a utilizar para el desarrollo

Tras haber seleccionado el tipo de aplicación y haber determinado los estilos arquitectónicos que más se ajustan al tipo de sistema que se va a construir, se tiene que decidir cómo construir los bloques que forman el sistema. Por ello, el siguiente paso es seleccionar las distintas tecnologías a usar. Estas tecnologías están limitadas por las restricciones de despliegue y las impuestas por el cliente. Hay que entender las tecnologías como los ladrillos que se utilizan para construir el sistema.

Tecnología (Del gr. τεχνολογία, de τεχνολόγος, de τέχνη, arte, y λόγος, tratado).

1. f. Conjunto de teorías y de técnicas que permiten el aprovechamiento práctico del conocimiento científico.
2. f. Tratado de los términos técnicos.
3. f. Lenguaje propio de una ciencia o de un arte.
4. f. Conjunto de los instrumentos y procedimientos industriales de un determinado sector o producto.(RAE 2011)

Partiendo de esto, a continuación se describen un conjunto de tecnologías importantes a tener en cuenta, para seleccionar cuáles son las más idóneas para aplicar en la implementación de la arquitectura a diseñar.

1.2.1 Administradores de Identidad Centralizados

Los administradores de identidad centralizados son aplicaciones que van a almacenar los datos de usuarios; estos sistemas se usan cuando tenemos muchos servicios en una empresa o institución y todos estos requieren de autenticación por parte de las personas que los usan. Permiten usar una sola identidad para acceder a todos los servicios. Intercambio de identidades entre proveedores.

Sistemas que implementan esta tecnología:

OpenSso: Permite la federación de identidades, gestión de la seguridad sobre los servicios que brinda, control de acceso, intercambio de atributos, manejo de sesión, interfaz de usuario para autenticación distribuida, manejo de idiomas, capa de servicios web para la interacción, además, es multiplataforma. Para el almacenamiento de datos se recomienda “Sun Java SystemDirectory Server 5.2, 6.0, 6.2 y 6.3”, el almacén de datos interno no es recomendable para muchos usuarios, en un entorno de desarrollo se requiere 1gb de RAM, en producción se requieren 4db. Se requieren 512mb en disco para los binarios del sistema y 7gb para archivos de registros; construido usando java. Tiene una versión libre para usar y una privativa para comercializar. Entre los estándares que soporta se encuentran: SAML, WS-Trust, WS-Security, WS-Authentication, WS-Federation. (CollabNet Inc.)

SimpleSAML: Permite federación de identidades, intercambio de atributos, manejo de sesión, interfaz de usuario para autenticación distribuida, internacionalización, es multiplataforma, no cuenta con una capa de servicios, esta liberado bajo licencia GPL. Soporta el estándar SAML. Fue creado usando PHP como lenguaje de programación. (SimpleSAMphp Project)

Java Open Single Sign-On Project Home (JOSSO): Permite el manejo de identidades, implementa el estándar SAML, maneja sesiones y sesiones únicas, capa de servicios; fue construido usando java. (González Oyuela 2010)

Spring Security: Permite la creación de una lista de control de acceso (ACL) por cada nombre de dominio, para pasar datos de autenticación se puede usar los encabezados http, manejo de sesiones únicas, soporte para OpenID; construido en java. (SpringSource)

Todos estos sistemas tienen una característica común y es que permiten definir los atributos asociados a las identidades, manejar las sesiones, así como sesiones únicas para todos los servicios.

1.2.2 Control de Acceso Centralizado

Los sistemas para el control de acceso centralizado, se encargan de manejar lo que puede hacer un usuario sobre un recurso determinado para un conjunto de sistemas. Le quitan a los sistemas la responsabilidad del control del acceso a sus recursos.

Sistemas que implementan esta tecnología:

OpenSSO: cumple con todos los requerimientos que necesita un sistema de control de acceso y de identidades.

Spring Security: es menos robusto y completo que OpenSSO, pero brinda las funcionalidades más comunes.

1.2.3 Servicios Web (Web services)

Muchas son las definiciones que existen sobre los Servicios Web, esto demuestra la complejidad para definirlos adecuadamente, de manera que se abarque todo lo que son e implican. Una posible sería, hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí, con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Estos servicios proporcionan mecanismos de comunicación estándares entre distintas aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Se necesita una arquitectura de referencia estándar, que permita proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que a la vez, sea posible su combinación para realizar operaciones complejas. (W3C 2010)

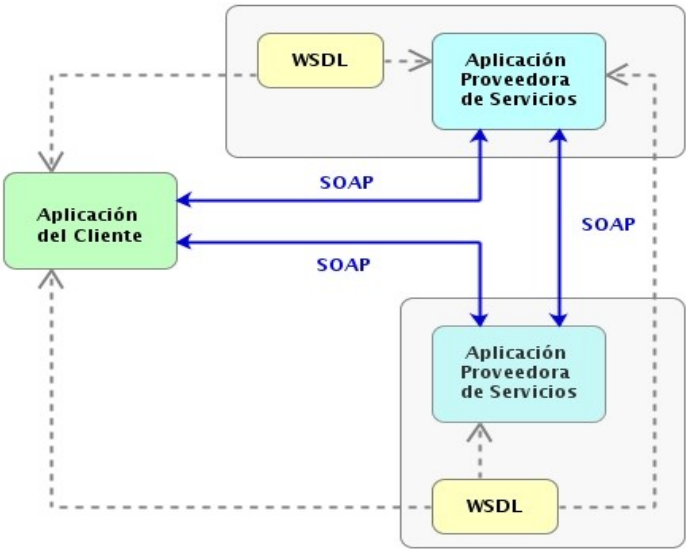

Figura 1: Los servicios web en funcionamiento

En la figura 1 una aplicación cliente (SC) consume servicios web que proveen otras dos aplicaciones mediante protocolo Simple Object Access Protocol (SOAP) (W3C). Cada aplicación proveedora de

servicios (SP) tiene asociado una descripción usando el Lenguaje de Descripción de Servicios Web (WSDL), el cual es necesario para garantizar el correcto entendimiento en una comunicación SOAP. Esta descripción en WSDL detalla cada una de las funciones, así como de los tipos de datos asociados a los parámetros y a los valores de retorno.

A menudo, cuando se denominan los servicios web, surgen los términos SOAP y REST (Representative State Transfer). REST es un estilo de arquitectura para generar aplicaciones cliente-servidor, mientras que SOAP es una especificación de protocolo para intercambiar datos entre dos extremos (Thomas Fielding 2000). A pesar de que REST es más simple de desarrollar que SOAP, que la curva de aprendizaje es pequeña y que es más conciso, pues no necesita de una capa de mensajería adicional; este tiene la desventaja que asume un modelo de comunicación punto a punto, que no es aplicable en entornos distribuidos donde los mensajes pueden atravesar uno o varios intermediarios.

CAPÍTULO 2: DESCRIPCIÓN Y ANÁLISIS DE LA PMSWL.

Una vez caracterizados los puntos esenciales para definir un mecanismo de integración, queda claro que debe definirse una solución para un conjunto de requisitos técnicos y operacionales, a partir del diseño de una arquitectura. Para llevar esto a cabo, primero se describe la concepción inicial de la plataforma; luego basado en un modelado del negocio se exponen las características que debe tener la misma según las exigencias del cliente. Finalmente se establecen qué componentes forman parte del sistema, cómo se relacionan entre ellos, y cómo mediante su interacción se llevan a cabo las funcionalidades especificadas.

2.1 Concepción inicial y definición de la plataforma.

El departamento SIMAYS, perteneciente al Grupo Técnico Nacional para la migración a software libre y código abierto, lleva a cabo los procesos de migración de sus clientes. Sin embargo, muchas de las tareas o actividades que se ejecutan para lograr el objetivo final tienen determinados inconvenientes, que muchas veces hacen ineficiente el proceso de migración. Se busca optimizar el tiempo durante el proceso de migración. Actualmente se pierde tiempo en:

- Analizar la información recopilada del levantamiento en las empresas de manera centralizada.
- La búsqueda de herramientas alternativas para la migración de las aplicaciones privativas.
- Falta de una planificación correcta en el proceso de migración.
- Medir el avance de los procesos de migración por empresa.
- Realizar informes para enviar al grupo ejecutivo, que muestren el avance de la migración, los informes en algunos casos van con información irreal, pues no se tienen los indicadores para medir la migración.
- Realizar búsquedas en internet para validar que el hardware detectado a través del sistema de gestión de inventarios OCSInventory, es compatible con la distribución de GNU/Linux Nova.

2.1.1 ¿Qué es?

La PMSWL es un sistema que está compuesto por un grupo de aplicaciones independientes que comparten información y funcionalidades entre sí.

2.1.2 Misión.

Integrar los subsistemas que componen la PMSWL. Comunicar los mismos garantizando la seguridad en el intercambio de datos. Brindar una interfaz única de fácil manejo y usabilidad para el usuario, que permita la gestión de los procesos de migración y el acceso a las distintas aplicaciones de la plataforma.

2.1.3 Visión.

Lograr que los componentes a integrar sean a su vez segregables e igualmente funcionales para terceros. Que esta plataforma pueda ser utilizada en cualquier proceso de migración a software libre y código abierto en Cuba y otras partes del mundo.

2.1.4 Solución propuesta.

Cuando se inicia un proceso de migración en una empresa, el administrador se autentica en la plataforma e inicia un nuevo proyecto de migración. Esto implica la generación automática de una nueva empresa en el sistema de planificación, control y seguimiento, con un proyecto de migración. El sistema LimeSurvey se encargará de automatizar las encuestas en la institución, ayudando a la identificación del tipo de usuario que participa en el proceso de migración y a clasificarlo según su grado de impacto en los procesos empresariales.

El flujo de evaluación propuesto por la metodología de migración, tiene como salida los datos recogidos por el sistema OCSInventory Client (o Everest en caso de no existir conectividad) sobre el levantamiento de una empresa determinada. La plataforma debe validar que el hardware detectado esté soportado por GNU/Linux Nova, para esto debe chequearse la base de datos del sistema de homologación. En el flujo de diseño se emplea entonces el sistema directorio de software, para buscar una alternativa libre a las aplicaciones detectadas provenientes del levantamiento del OCSInventory. La plataforma evaluará a petición del usuario, a través del sistema de métricas en qué por ciento de

cumplimiento está el proceso de migración, definido en las actividades del plan de migración correspondiente a cada empresa. Durante todo el proceso de migración, el Centro de Gestión de COMOs Online servirá de apoyo.

2.2 Negocio y Requisitos.

La metodología de desarrollo ágil SXP establece que en la fase de definición han de identificarse los requisitos claves del software. Cuenta con tres tareas principales: ingeniería de sistemas o de información, planificación del proyecto y análisis de los requisitos. (Meneses Abad et al. 2009)

Los requisitos que se capturan son priorizados y expresados en una Lista de Reserva del Producto (LRP) la cual define el trabajo que se va a realizar en el proyecto. En la confección de la LRP participan el Jefe de proyecto, el gerente, el cliente y el equipo de proyecto. El objetivo es asegurar que el producto definido al terminar la lista es el más correcto, útil y competitivo posible.

2.2.1 Modelo de Negocio.

Si se considera la migración a software libre en una entidad como el proceso de negocio, puede identificarse como actor a la *entidad cliente,* que puede ser la empresa o institución que se desea migrar. Esta se beneficia de las actividades que realizan el *líder de equipo de migración* quien organiza y dirige el proceso en la entidad cliente, el *especialista de migración* que ejecuta las actividades propias de las etapas de migración, el *especialista de redes y servicios telemáticos* que analiza las características de la red de la institución y despliega los servicios telemáticos de la misma y el *especialista funcional* que lleva a cabo tareas de apoyo al proceso de migración. Para comprender mejor el negocio se ofrece a continuación el diagrama de historias de usuario correspondiente.

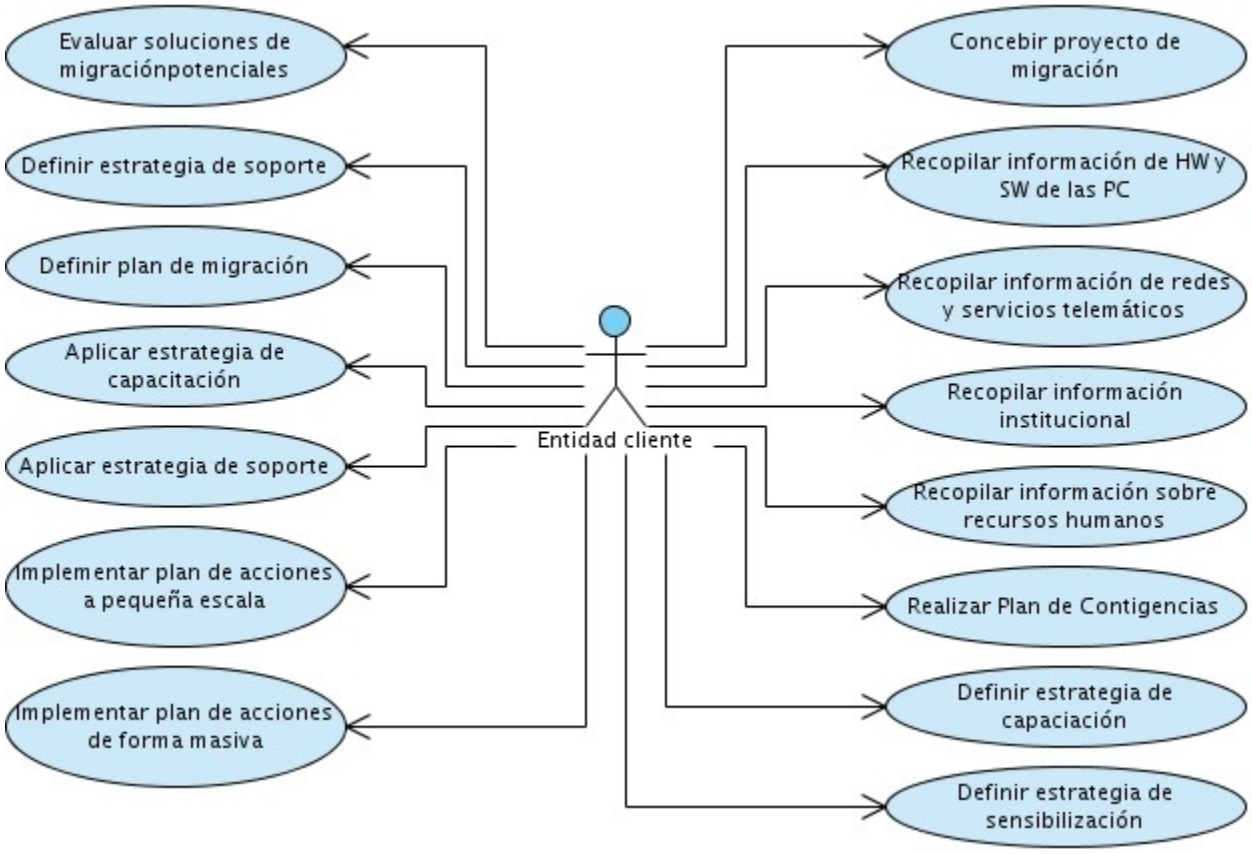

Figura 2: Diagrama de Historias de Usuario del Negocio

2.2.2 Requisitos para la integración.

Durante la reunión con el cliente, se llegó a un acuerdo con el equipo de proyecto, sobre lo que debería ser la integración de los componentes de la PMSWL a partir de la definición de los siguientes requisitos:

1. Crear interfaz sencilla e intuitiva desde la cual se acceda a todas las funcionalidades de los sistemas que componen la plataforma.
2. Los usuarios con conocimiento medio y avanzado requerirán de 1 semana y 3 días respectivamente para ser 100% productivos operando la plataforma.
3. Los componentes para la integración deben funcionar sobre plataforma web.
4. Todos los sistemas componentes de la plataforma intercambiarán información mediante servicios web.

5. La información de las credenciales deberá ser común para todos los sistemas al igual que los datos de cada persona.
6. La plataforma podrá funcionar con los sistemas distribuidos en una red de buenas prestaciones.
7. Los componentes desarrollados para la integración deberán ser implementados en lenguaje PHP.
8. La entrega de la primera versión funcional de la plataforma deberá ser antes del 30 de mayo de 2011.
9. Cada acción que se realice sobre los sistemas de la plataforma deberá ser registrado quién y cuándo lo hizo.
10. Se estima 1 error / MLC como mínimo.
11. La plataforma deberá funcionar 8 horas al día, se prevé un acceso semanal para mantenimiento.
12. La plataforma debe admitir el acceso promedio de 20 usuarios simultáneamente.
13. La interfaz de la plataforma debe estar acorde con la identidad del departamento SIMAYS.
14. La plataforma será liberada bajo licencia establecida por la UCI.
15. Deben utilizarse tecnologías libres para el desarrollo de la plataforma.
16. La plataforma deberá funcionar sobre el sistema operativo GNU/Linux Nova.

2.3 Descripción de la arquitectura.

2.3.1 Herramientas asociadas al desarrollo del sistema.

Entornos integrados de desarrollo (IDE): NetBeans

Se utiliza el NetBeans en su versión 6.9 para desarrollar y depurar el código de los módulos de todos los subsistemas de la plataforma.

Marcos de trabajo (Frameworks) que soportarán el desarrollo: Symfony y Jquery

Se utilizará el framework symfony para la programación de la lógica de las aplicaciones de la plataforma. En el caso del framework y librería de JavaScript Jquery se empleará para la creación de los prototipos de interfaces de usuario de los diferentes módulos de gestión de la plataforma.

Gestor de bases de datos: PostgreSQL

Se empleará PostgreSQL por las ventajas que posee ante el desarrollo de este sistema. Es multiplataforma, por lo que funciona perfectamente en los sistemas operativos más populares en la actualidad; cuestión indispensable y necesaria para poder desplegar este sistema en cualquier sistema operativo. Además, tiene soporte nativo para muchos lenguajes de programación, entre ellos PHP. También permite todo tipo de consultas a bases de datos. Lenguaje procedimental propio denominado PL/PgSQL. Brinda soporte multiusuario, para integridad referencial, transacciones y control de concurrencia, SSL, presenta tipos de datos del estándar SQL, así como soporte para tipos de datos creados por el usuario.

Lenguaje de programación: PHP y Javascript

Se elige PHP para la programación de las funcionalidades de cada subsistema, permitiendo construir un sistema web dinámico. Javascript se utilizará para manipular las interfaces del lado del cliente.

Protocolos de comunicación: SOAP, HTTP y HTTPS

Se empleará el protocolo HTTP por ser el protocolo estándar destinado a la web y es empleado en la mayoría de las aplicaciones de este tipo. El protocolo HTTPS será empleado en la comunicación entre los componentes de la plataforma, en los momentos que se requiera intercambiar información sensible entre ellos, como usuarios y contraseñas. Ambos protocolos serán los empleados en la comunicación. El protocolo SOAP cumple sus funciones específicamente sobre los servicios web, en la capa de mensajería de la arquitectura de un servicio web, para la codificación de los mensajes XML y que sean entendibles entre las aplicaciones de la plataforma.

Ingeniería de software asistida por computadoras (CASE): Visual Paradigm for UML

Se empleará Visual Paradigm for UML (VP-UML) como herramienta de diseño UML que permite modelar los artefactos generados durante todo el desarrollo del proyecto; así como para diseñar los prototipos de interfaz de usuario. Esta herramienta aunque no es libre, constituye la mejor opción para realizar estas tareas y fue utilizada en versión para GNU/Linux.

Sistema de control de versiones: Subversion

Este importante sistema de control de versiones permite tener un control de los artefactos y códigos de las aplicaciones durante todo el desarrollo del proyecto hasta obtener el producto final. Y como herramienta para conectarse a Subversion y realizar operaciones sobre el repositorio se elige el Rapidsvn, por su facilidad de uso.

2.3.2 Estructuración de los componentes.

Fundamentación de los estilos arquitectónicos utilizados:

Como sucede generalmente, una arquitectura combina varios estilos arquitectónicos con el objetivo de utilizar las ventajas de cada uno de ellos para lograr una mejor solución. En el caso de la PMSWL, se propone la utilización de SOA y Presentación Desacoplada. Aunque la arquitectura propuesta tenga en cuenta elementos comunes a otros estilos, son los dos mencionados anteriormente los que tienen mayor presencia.

¿Por qué SOA?

La necesidad de una interfaz, desde la cual se pueda acceder a todas las funcionalidades de la plataforma y de que todos sus sistemas componentes intercambiarán información mediante servicios web, evidencia la utilidad de aplicar SOA.

¿Por qué Presentación Desacoplada?

El uso de este estilo es evidente al querer simplificar el código de las interfaces con el objetivo de facilitar el proceso de desarrollo y que la plataforma pueda tener varias interfaces para una misma lógica de negocio.

En la siguiente figura se muestra cómo quedarían las relaciones entre los componentes de la PMSWL:

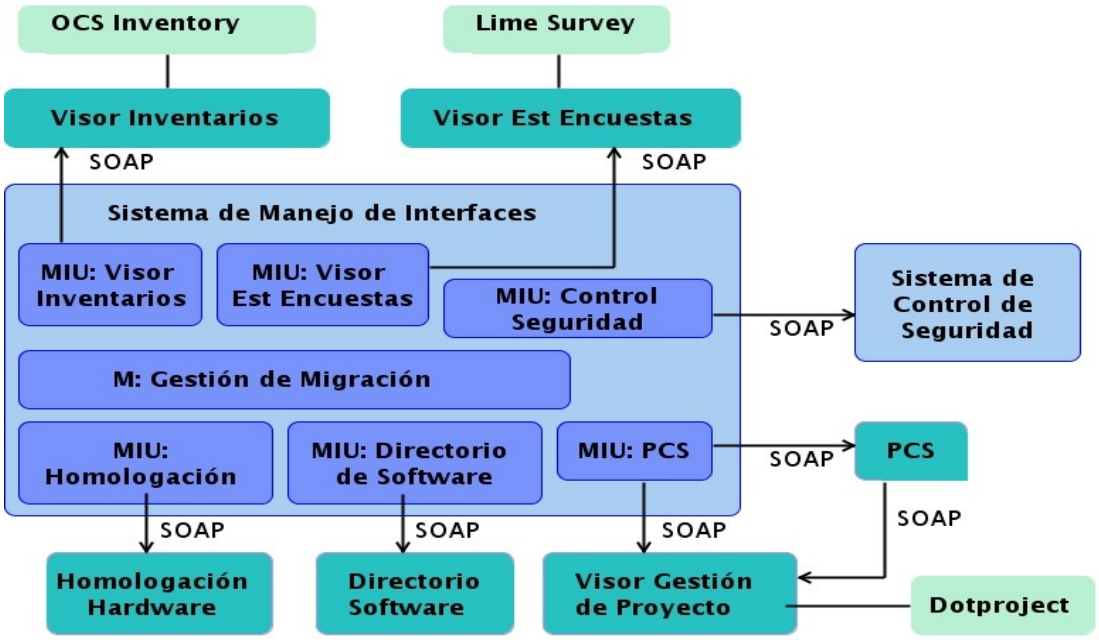

Figura 3: Estructura de los componentes de la PMSWL

Descripción de los componentes:

- *OCS Inventory:* es el sistema encargado de realizar el inventario de hardware y software, este es desplegado en la entidad a migrar y la base de datos resultante es copiada a un servidor de la plataforma, encargado de almacenar todas las bases de datos generadas por este software, asociándolas a la entidad correspondiente.
- *Lime Survey*: es el sistema que gestiona las encuestas empleadas en el levantamiento de información según la Metodología de Migración. Al igual que el OCS Inventory, la base de datos generada en cada entidad es copiada a un servidor.
- *Dotproject:* es el sistema que permite la gestión de proyectos de migración de cada entidad.
- *Visor Inventarios, Visor Est. Encuestas y Visor Gestión de Proyecto*: Constituyen una capa de servicios para el OCS Inventory, el Lime Survey y el Dotproject respectivamente.
- *PCS:* es el sistema de planificación, control y seguimiento de los procesos de migración, que permite medir sus avances a partir de indicadores. Su funcionamiento se complementa con el Visor Gestión de Proyectos para determinar la etapa en que se encuentra un proceso.

- *Directorio de Software:* es el sistema para buscar alternativas de software libre al software privativo que, según el OCS Inventory, fue detectado en determinada entidad.
- *Homologación de Hardware:* Sistema para validar la compatibilidad del hardware detectado con plataforma GNU/Linux Nova.
- *Sistema de Control de Seguridad (SCS):* el sistema encargado de proveer las identidades y el control de acceso de manera centralizada.
- *Sistema para el manejo de interfaces (SMI):* es el sistema encargado de gestionar las interfaces correspondientes a cada sistema anteriormente descrito, usando para cada uno un módulo de interfaz de usuario (MIU), además cuenta con un módulo (M) destinado a la gestión del proceso de migración en su totalidad, que no responde a un sistema en específico, pero depende de todos ellos para su funcionamiento.

Descripción de las relaciones más significativas entre componentes:
Para comprender cómo se establecen las relaciones entre un MIU y su sistema correspondiente se muestra la siguiente figura:

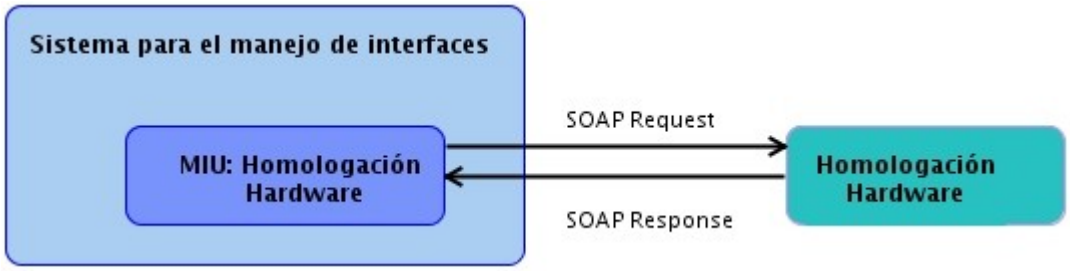

Figura 4: Ejemplo de interacción entre un sistema y su MIU asociado

La comunicación en SOAP siempre tiene un mensaje de solicitud (SOAP Request), donde se especifica qué servicio se quiere consumir y se pasan los parámetros que requiere el mismo. El sistema publicador del servicio solicitado enviará una respuesta (SOAP Response), que puede ser información que se necesite retornar, mensajes de error o confirmación.

En la PMSWL todo mensaje de solicitud va acompañado de un identificador de sesión asociado al usuario y las credenciales del sistema de manejo de interfaces de usuario.

El manejo de la seguridad en la PMSWL puede entenderse a partir de la siguiente figura, que modela un ejemplo de interacción entre un sistema, su MIU asociado y el SCS:

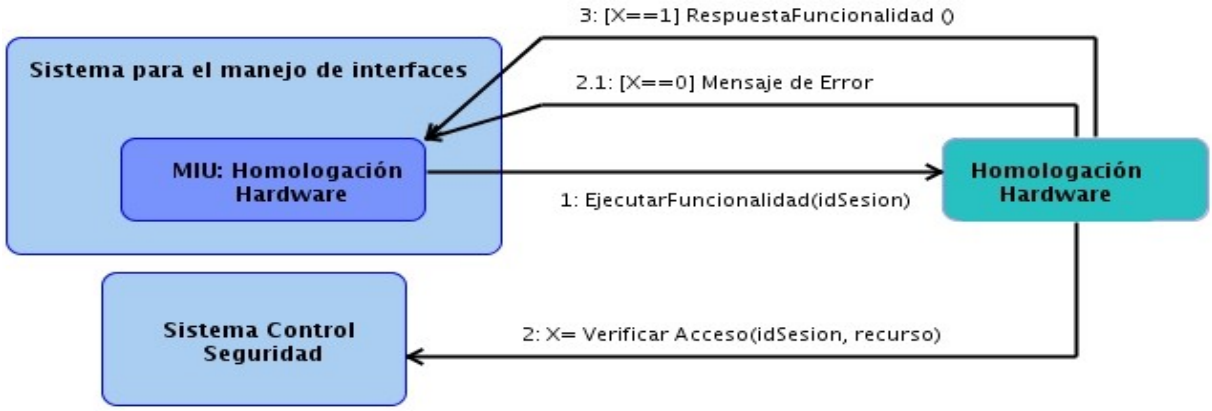

Figura 5: Ejemplo de funcionamiento de la seguridad en la PMSWL

Cuando se intente consumir un servicio web publicado por uno de los sistemas, este último le preguntará al SCS si permite o no la ejecución de dicho servicio.

2.3.3 Estructura de la distribución física del sistema.

El uso del estilo SOA en esta arquitectura permite que la PMSWL pueda funcionar con sus sistemas segregados o centralizados, en dependencia del entorno en el que se despliegue la misma, y los recursos que existan para ello. En la figura 6 puede apreciarse el diagrama de despliegue en el que se muestran dos servidores de aplicaciones donde se encuentran alojados los sistemas proveedores de servicios, estos servidores interactúan con un servidor de bases de datos, la cantidad de servidores de aplicaciones y de bases de datos está en dependencia del nivel de segregación que se quiera implementar. Las PC clientes interactúan directamente con un servidor donde se encuentra alojado solamente el SMI que interactúa mediante SOAP con los servidores de aplicaciones mencionados. Los servidores de aplicaciones proveedoras de servicios y de bases de datos deberán estar en una red a la

cual sólo se tenga acceso desde el servidor con el SMI, de esta forma se contribuye a la seguridad de la PMSWL.

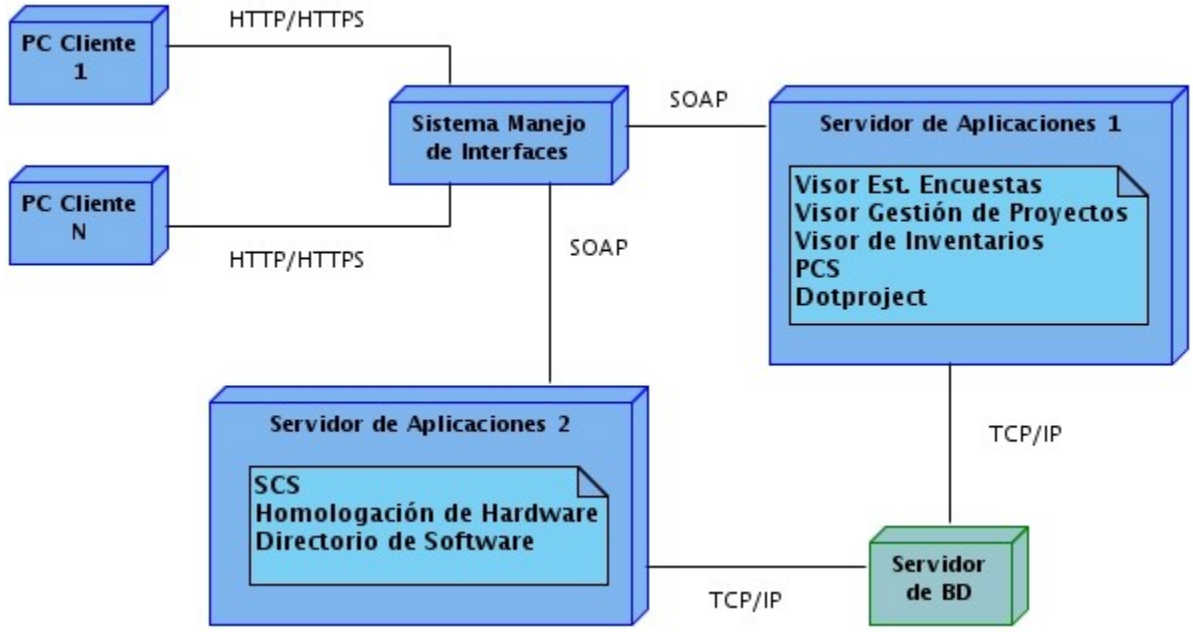

Figura 6: Diagrama de despliegue de la PMSWL.

CAPÍTULO 3: DESARROLLO ÁGIL DEL SCS.

A partir del diseño de la arquitectura de la PMSWL, se decide que para llevar a cabo la integración, es necesario el desarrollo de dos sistemas: SCS y SMI. Este capítulo está dedicado al desarrollo del SCS, para ello se definen las funcionalidades del mismo que se expresan de forma priorizada en una lista de reserva del producto (LRP); luego, estas son especificadas a través de las historias de usuario (HU) con sus respectivas tareas de ingeniería llevadas a cabo durante la implementación; se muestran tanto la estructura lógica como física del sistema en cuestión mediante el modelo de diseño y el diagrama de componentes respectivamente, además de mostrar el modelo de datos asociado y finalmente se brinda un manual de configuración, para permitir a los administradores de la plataforma poner el sistema en funcionamiento.

3.1 Lista de Reserva del Producto.

Durante la captura de requisitos se confecciona la LRP, en la cual se definen las funcionalidades que tendrá el producto en forma de requisitos técnicos y de negocio. Es una lista priorizada y garantiza la organización de los requisitos funcionales y no funcionales, a partir de la prioridad que tenga para el desarrollo del sistema, de igual forma facilita el trabajo al confeccionar las HU.

En el caso del SCS se planificó su desarrollo en una única iteración y todos los requisitos identificados son de prioridad muy alta, dado que los otros sistemas en construcción dependen del desarrollo de las funcionalidades que se exponen a continuación:

Asignado a	Item	Descripción	Estimación (en horas)	Estimado por
Jailen García	1	Insertar usuario	20	Abel García
	2	Actualizar usuario	3	
	3	Mostrar usuario	3	
	4	Activar usuario	3	

	5	Desactivar usuario	3	
	5	Insertar Sistema	3	
	7	Actualizar Sistema	3	
	8	Eliminar Sistema	3	
	9	Mostrar Sistema	3	
	10	Insertar recurso de sistema	3	
	11	Actualizar recurso de sistema	3	
	12	Eliminar recurso de sistema	3	
	13	Mostrar recurso de sistema	3	
	14	Iniciar sesión	3	
	15	Verificar si usuario tiene permiso de realizar acción	3	
	16	Finalizar sesión	3	
	17	Verificar si una sesión esta activa.	3	
	18	Insertar regla	3	
	19	Actualizar regla	3	
	20	Eliminar regla	3	
	21	Mostrar regla	3	

3.2 Historias de usuario.

Las HU son la técnica utilizada en SXP para especificar los requisitos del software, lo que equivaldría a los casos de uso en el proceso unificado. Las historias de usuario guían la construcción de las pruebas de aceptación y son utilizadas para estimar tiempos de desarrollo. En este sentido, sólo proveen detalles suficientes para hacer una estimación razonable del tiempo que llevará implementarlas. En el momento de implementar una historia de usuario, se debe detallar a través de la comunicación con el cliente, siendo estas son la base para las pruebas funcionales. (Penalver Romero 2008)

Las tareas de ingeniería definen cada una de las actividades asociadas a las HU y permiten organizar el proceso de implementación, así como conocer el grado de complejidad de cada HU, teniendo en cuenta la cantidad de tareas asociadas. En este epígrafe se describen las HU del SCS.

3.2.1 Gestionar usuario.

Historia de Usuario	
Número: 11	**Nombre Historia de Usuario:** Gestionar usuario
Modificación de Historia de Usuario Número: 1	
Usuario: Jailen García González	**Iteración Asignada:** 1
Prioridad en Negocio: Muy alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2.5 semanas
Descripción: Se gestionan las identidades de los usuarios que interactúan con todos los sistemas de la plataforma, permitiendo *adicionar, mostrar, actualizar, activar y desactivar* cada una de las identidades.	
Observaciones: • Para **adicionar** una identidad (usuario), deben completarse los siguientes campos: nombre, primer apellido, segundo apellido, dirección de correo, identificador del país al que pertenece, sexo (Masculino, Femenino), nombre de usuario, contraseña, activo (sí o no). ◦ Las funcionalidades **activar** y **desactivar** se usan para cambiar el valor del campo activo. ◦ Cuando se adiciona un usuario el campo activo se encuentra marcado por defecto. La contraseña tiene que ser introducida dos veces para verificar coincidencia. ◦ Al campo contraseña se le aplica la función resumen SHA1. • Para **actualizar** se pueden modificar todos los campos anteriores, excepto el nombre de usuario y el identificador que se auto-genera cuando se adiciona. • Para mostrar los usuarios, se debe tener un paginado de manera que la lista visualizada no sea demasiado extensa y también debe poderse mostrar los detalles de un usuario en específico.	

Prototipo de interfaz:

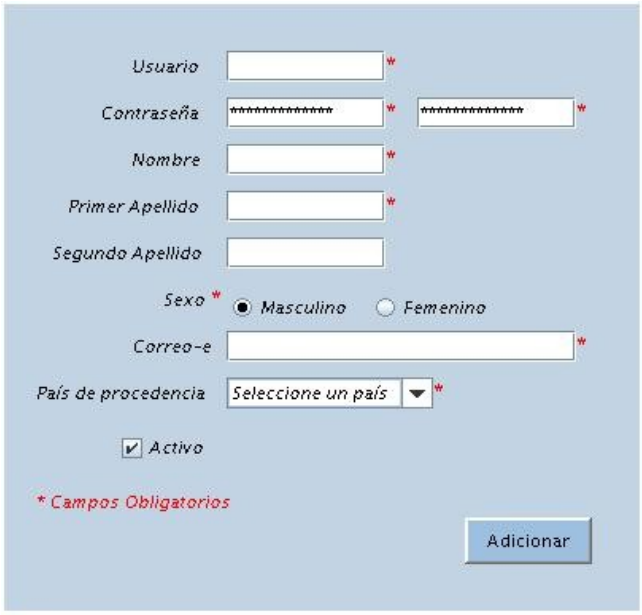

Figura 8: Prototipo de "Adicionar usuario".

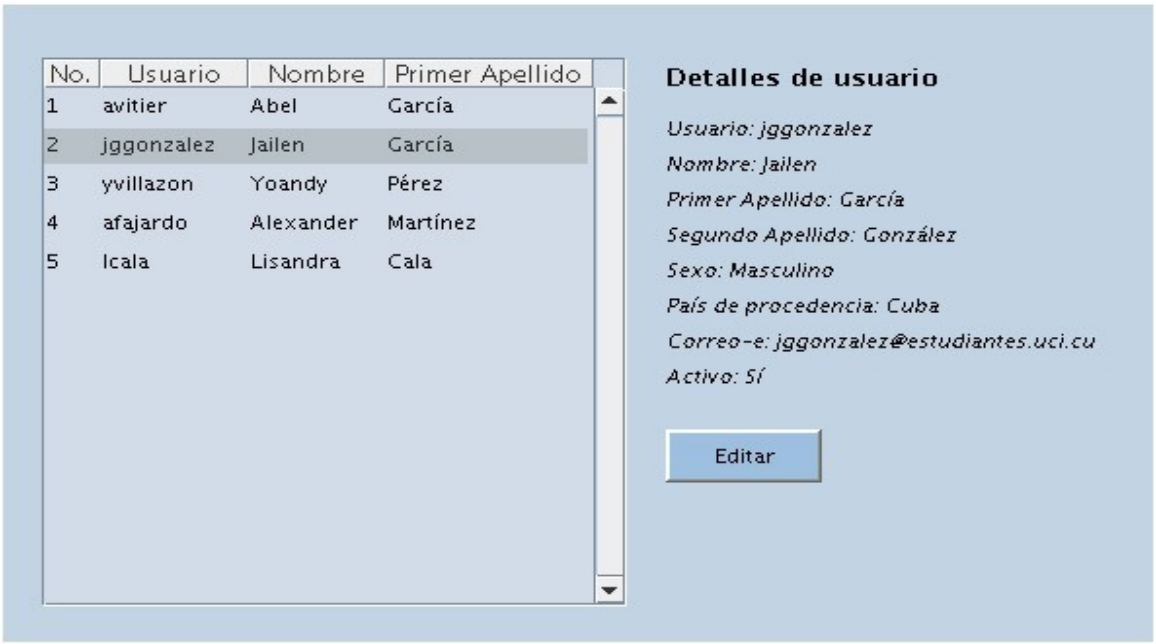

Figura 9: Prototipo de "Mostrar Usuarios".

3.2.2 Gestionar sistema.

Historia de Usuario	
Número: 13	**Nombre Historia de Usuario:** Gestionar sistema
Modificación de Historia de Usuario Número: 1	
Usuario: Jailen García González	**Iteración Asignada:** 1
Prioridad en Negocio: Muy alta	**Puntos Estimados:** 1 semana
Riesgo en Desarrollo: Medio	**Puntos Reales:** 1 semana
Descripción: Se gestionan los datos de los sistemas que podrán utilizar los servicios web brindados por la aplicación. Se permite *adicionar, mostrar, actualizar y eliminar sistemas.*	
Observaciones: • Para adicionar un sistema se necesita un nombre (sin caracteres de espacio, no nulo y único), un nombre descriptivo (puede tener el nombre que se le muestra a los usuarios), la url del fichero wsdl correspondiente al sistema (puede ser nula, tiene que ser una url valida según normas internacionales), la llave publica del sistema (puede ser nula), un nombre de usuario (no puede ser nulo, sin caracteres de espacio y único) y una contraseña (se le aplica función resumen SHA1). • Para **actualizar** se pueden modificar todos los campos, excepto el id auto-generado. • El **mostrar** consiste en listar todos los sistemas. • Para **eliminar** se selecciona el sistema y se confirma la eliminación. Eliminar un sistema implica eliminar todos los recursos asociados a este.	

Prototipo de interfaz:

Figura 10 Prototipo de "Adicionar sistema".

Figura 11: Prototipo de "Mostrar sistema".

3.2.3 Gestionar recurso de sistema.

Historia de Usuario	
Número: 14	**Nombre Historia de Usuario:** Gestionar recurso de sistema
Modificación de Historia de Usuario Número: 1	
Usuario: Jailen García González	**Iteración Asignada:** 1
Prioridad en Negocio: Muy alta	**Puntos Estimados:** 1 semana
Riesgo en Desarrollo: Medio.	**Puntos Reales:** 1.5 semanas
Descripción: Se gestionan los recursos que representan funcionalidades de los sistemas componentes de la plataforma. Se permite *adicionar, mostrar, actualizar y eliminar* recursos.	
Observaciones: • Para adicionar un recurso se necesita: uri (cadena que representa un recurso, como por ejemplo, un documento o un servicio web, no puede ser nula y es única), descripción (puede ser nula) y el identificador del sistema al que está asociado dicho recurso. • Para actualizar se pueden modificar todos los campos, pero no el id auto-generado. • El mostrar consiste en listar todos los recursos asociados a un sistema. • Para eliminar se selecciona el recurso y se confirma la eliminación.	
Prototipo de interfaz:	

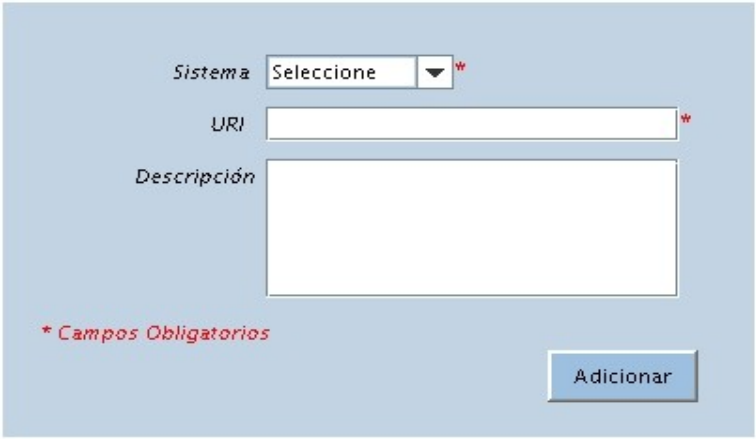

Figura 12: Prototipo de "Adicionar recurso de sistema".

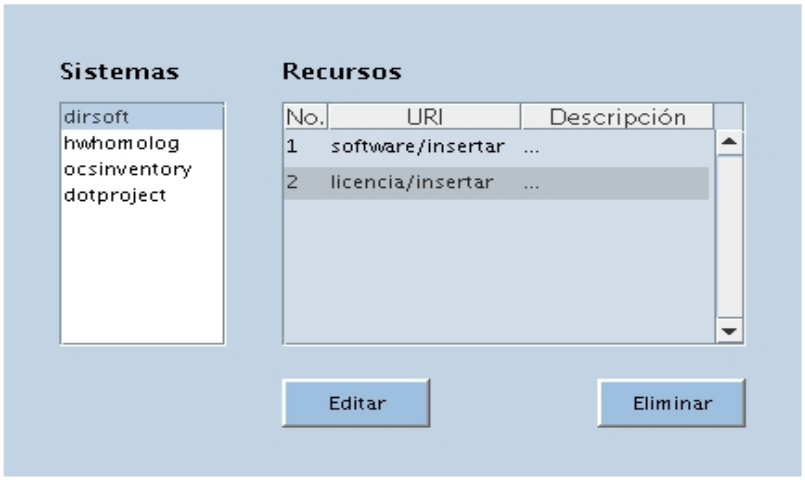

Figura 13: Prototipo de "Mostrar recurso de sistema".

3.2.4 Manejo de sesiones.

	Historia de Usuario
Número: 15	**Nombre Historia de Usuario:** Manejo sesiones
Modificación de Historia de Usuario Número: 1	
Usuario: Jailen García González	**Iteración Asignada:** 1
Prioridad en Negocio: Muy alta	**Puntos Estimados:** 2 días
Riesgo en Desarrollo: Bajo	**Puntos Reales:** 3 días
Descripción: Se manejan las sesiones de los usuarios, una sesión representa un usuario autenticado. Se permite *Iniciar una sesión, finalizar sesión, verificar si una sesión esta activa, verificar si el usuario tiene permiso de realizar una acción.*	
Observaciones: • Para iniciar una sesión se requiere un usuario y una contraseña, se verificará que el usuario dado exista en la base de datos y que la contraseña almacenada coincida con la que se provee. Si no existe, se lanzará un error, en caso de existir, se deberá verificar si ya tiene una sesión iniciada y válida*, en cuyo caso se retornará el identificador de dicha sesión. De no tener sesión iniciada se le creará una, para esto se generará una cadena que sirva como id, se almacenará la fecha de creación y el id del usuario al que representa este objeto sesión.	

- Para finalizar una sesión solo ser requiere eliminar el registro de la sesión en la base de datos.
- Para verificar si una sesión esta activa se deberá chequear que sea válida.
- Para verificar si usuario tiene permiso de realizar una acción se necesita el recurso, el id de la sesión y la acción a realizar.

*Una sesión es válida si no h sión no es válida deberá ser eliminada automáticamente

*Existe un administrador que

Prototipo de interfaz:

Figura 14: Prototipo de "Iniciar sesión".

3.2.6 Gestionar reglas.

	Historia de Usuario
Número: 16	**Nombre Historia de Usuario:** Gestionar reglas
Modificación de Historia de Usuario Número: 1	
Usuario: Jailen García González	**Iteración Asignada:** 1
Prioridad en Negocio: Muy alta	**Puntos Estimados:** 1.5 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2 semanas
Descripción: Una regla nos dice si un conjunto de usuarios puede o no realizar un conjunto de acciones sobre determinados recursos. Se necesita poder *adicionar, mostrar, actualizar y eliminar* reglas.	

Observaciones:

- Para adicionar una regla se necesita conocer el tipo de regla (Permitir o Denegar), el identificador de todos los usuarios que participan en la regla, así como el id de las acciones y recursos involucrados.
- En la actualización de una regla se pueden adicionar o quitar usuarios y recursos participantes; incluso cambiar el tipo de la regla.
- Para mostrar las reglas en los usuarios, se debe mostrar el nombre de los mismos, de igual manera los demás campos deben mostrar elementos descriptivos.
- Para eliminar una regla sólo se deben borrar los registros que guarden información referente a la misma.

Prototipo de interfaz:

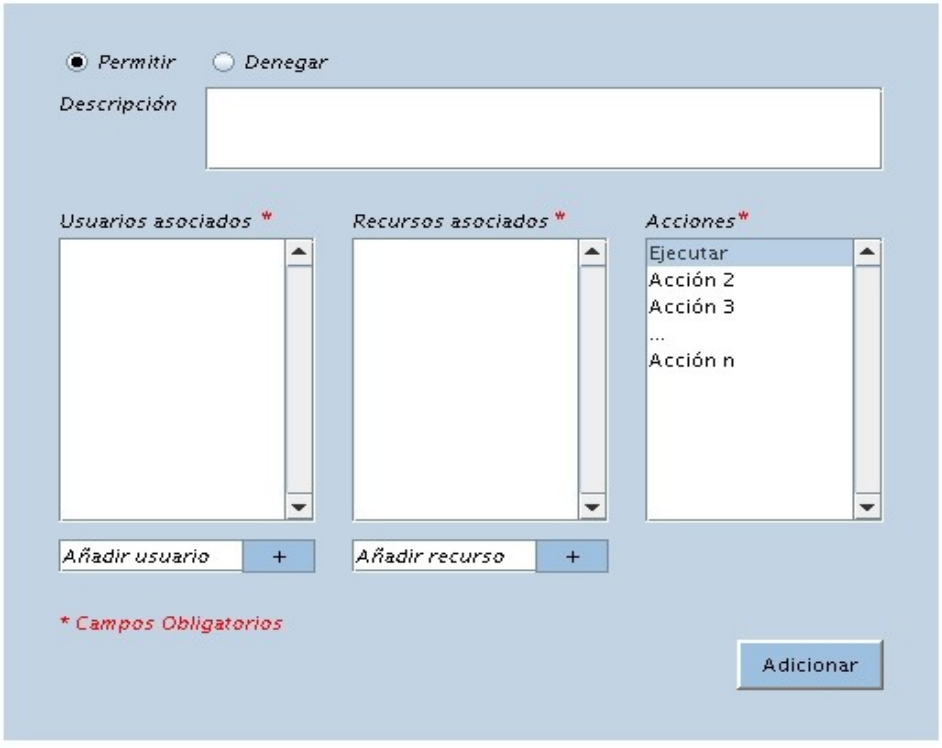

Figura 15: Prototipo de "Adicionar regla".

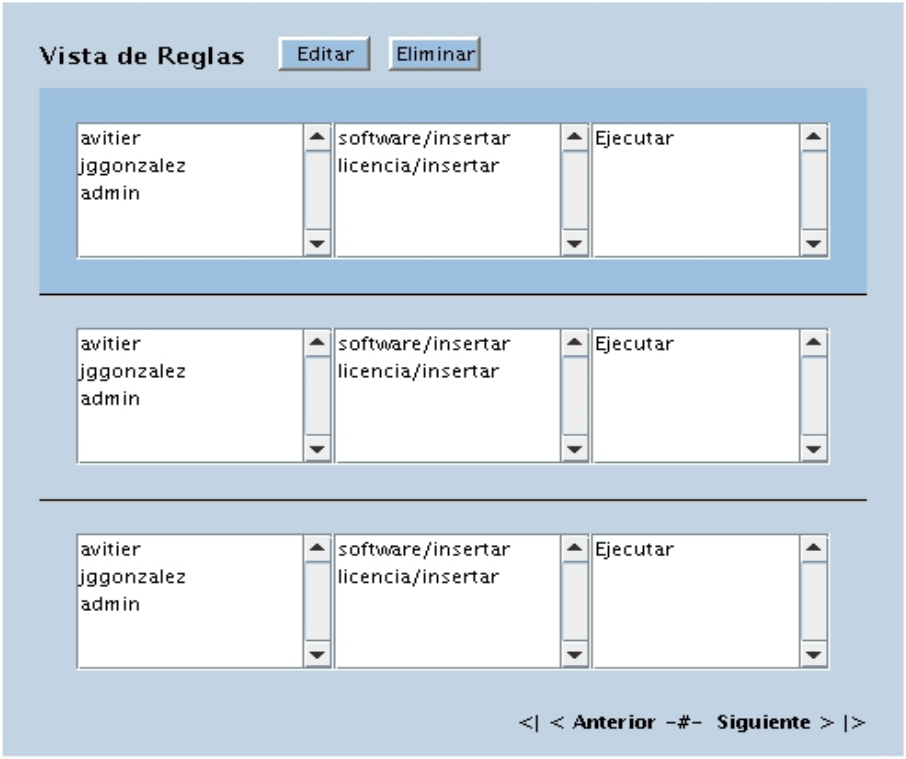

Figura 16: Prototipo de "Mostrar reglas".

3.3 Modelo de diseño

En la metodología SXP no se enfatiza en la definición temprana de una arquitectura estable para el sistema, sino que esta se asume de forma evolutiva. Para solventar los inconvenientes que pudiera generar el no contar con dicha arquitectura, se utiliza el diseño con metáforas que da como resultado el modelo de diseño.

El diseño del SCS, como parte de la PMSWL, debe acoplarse a la arquitectura de la misma (SOA + Presentación desacoplada) para poder establecerse la integración, sin embargo, este sistema a lo interno puede tener características estructurales afines a otro estilo arquitectónico, en este caso se optó por una arquitectura en capas, debido a que se desea exponer la lógica de negocio mediante interfaces de servicios, además, el SCS debe soportar distintos tipos de gestores de bases de datos. Este estilo

arquitectónico permite la realización de actualizaciones en el interior de las capas, sin que esto afecte al resto del sistema y el hecho de distribuir las capas en distintos niveles físicos, mejora la escalabilidad y la tolerancia a fallos.

En la figura 17 se muestra cómo fue diseñado este sistema en términos de clases y paquetes:

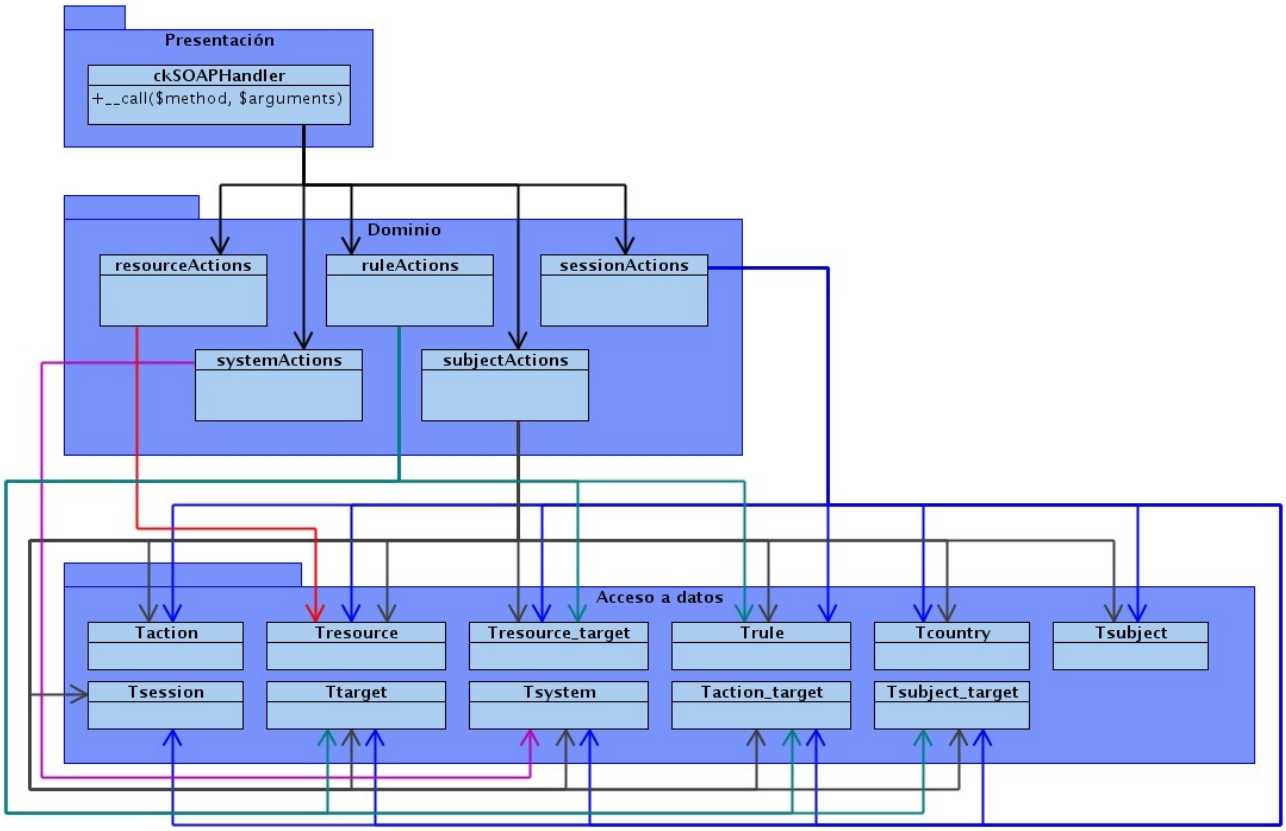

Figura 17: Diagrama de clases del diseño del SCS.

Puede apreciarse que las clases se encuentran agrupadas en tres capas lógicas, siguiendo un estilo N-Capas que no entra en conflicto con los demás estilos utilizados en esta arquitectura. La capa presentación contiene la clase *ckSOAPHandler* cuya función es manejar el protocolo SOAP, traduciendo funcionalidades en servicios web. En la capa dominio se encuentran las clases que controlan la lógica del negocio, correspondiendo una clase a cada una de las HU explicadas con anterioridad, estas clases, que a continuación se detallan, se relacionan con las clases de la capa acceso a datos, lo que permite abstraerse del sistema gestor de bases de datos que se utilice.

3.3.1 Clase resourceActions.

Método	Descripción	Parámetros	Valor de Retorno
preExecute	Controla el acceso del sistema y los usuarios a los servicios.	-	-
addResource	Añade un nuevo recurso al sistema.	$uri : string $description : string $idSystem : int	int (id del recurso añadido).
updateResource	Modifica un recurso existente en el sistema.	$id : int $uri : string $description : string $idSystem : int	bool (true si modificó).
getResource	Obtiene un recurso del sistema a partir de su identificador.	$id : int	Resource (el recurso obtenido).
listResource	Muestra una lista de recursos utilizando paginado, por lo que devuelve un conjunto de recursos a mostrar de la cantidad total.	$idSystem : int $uriSystem : string $pagina : int $cantidadPorPagina : int	ResourceMeta (lista de recursos a mostrar y cantidad total de recursos).
deleteResource	Elimina recursos del sistema a partir de una lista de identificadores.	$ids : int[]	bool (true si se eliminaron).

3.3.2 Clase sessionActions.

Método	Descripción	Parámetros	Valor de Retorno
preExecute	Controla el acceso del sistema y los usuarios a los servicios.	-	-

authentication	Inicia la sesión a partir de un usuario y contraseña, si la sesión existe retorna el id de la misma.	$user : string $passwd : string	string (id de la sesión).
tokenValidation	Valida si la sesión no ha expirado.	$token : string	bool (true si no ha expirado).
logout	Cierra la sesión.	$token : string	bool (true si procede).
authorize	Valida la ejecución de una acción sobre un usuario a partir del token de sesión.	$sessionToken : string $resourceUri : string $actionDescription : string	bool (true si procede).

3.3.3 Clase systemActions.

Método	Descripción	Parámetros	Valor de Retorno
addSystem	Añade un nuevo sistema a la plataforma.	$name : string $descriptiveName : string $urlWsdl : string $publicKeyUrl : string $user : string $passwd : string	int (id del sistema insertado).
getSystem	Obtiene un sistema a partir de su identificador.	$id : int	System (el sistema obtenido)
listSystems	Muestra los sistemas de la plataforma con sus respectivos datos.	-	System[] (lista de sistemas obtenidos)
updateSystem	Modifica un sistema existente.	$id : int	bool (true si

		$name : string $descriptiveName : string $urlWsdl : string $publicKeyUrl : string $user : string $passwd : string	modificó).
deleteSystem	Elimina sistemas a partir de un conjunto de identificadores.	$ids : int[]	bool (true si se eliminaro)

3.3.4 Clase subjectActions.

Método	Descripción	Parámetros	Valor de Retorno
addSubject	Adiciona un nuevo usuario a la plataforma.	$name : string $lastName : string $secondLastName : string $eMail : string $idCountry : int $sex : string $user : string $passwd : string $active : int	int (id del usuario insertado)
updateSubject	Modifica un usuario existente.	$id : int $name : string	bool (true si modificó)

		$lastName : string $secondLastName : string $eMail : string $idCountry : int $sex : string $user : string $passwd : string $active : int	
getSubject	Obtiene un usuario a partir de un identificador.	$id : int	Subject (usuario obtenido)
getPermissions	Devuelve una lista de permisos para un usuario a partir de su nombre.	$user : string	Permissions[] (lista de permisos)
listSubject	Muestra una lista de usuarios utilizando paginado, por lo que devuelve un conjunto de usuarios a mostrar de la cantidad total.	$nombre : string $usuario : string $pagina : int $cantidadPorPagina : int	SubjectMeta (lista de usuarios a mostrar y cantidad total de usuarios).
listCountries	Devuelve una lista de países.	-	Country[] (lista de países)
deleteSubjects	Elimina usuarios a partir de un conjunto de identificadores	$ids : int[]	bool (true si se eliminaron)

3.3.5 Clase ruleActions.

Método	Descripción	Parámetros	Valor de Retorno
preExecute	Controla el acceso del sistema y los usuarios a los servicios.	-	-
addRule	Adiciona una nueva regla al sistema.	$name : string $description: string $resources : int[] $users: int[]	int (id de la regla insertada)
updateRule	Modifica una regla existente.	$id : int $name : string $description: string $resources : int[] $users: int[]	int (id de la regla modificada)
getRule	Obtiene una regla a partir de su identificador.	$id : int	RuleDetails (Regla detallada)
listRule	Muestra una lista de relgas utilizando paginado, por lo que devuelve un conjunto de reglas a mostrar de la cantidad total.	$usuario : string $nombreregla : string $recurso : string $pagina : int $cantidadPorPagina : int	RuleMeta (lista de reglas a mostrar y cantidad total de reglas)
deleteRule	Elimina reglas a partir de un conjunto de identificadores.	$ids : int[]	Bool (true si se eliminaron)

3.4 Estructura de componentes.

Los componentes o ficheros correspondientes al SCS mantienen una organización física estrechamente relacionada con la estructura lógica de este sistema. Existe una carpeta dedicada a los módulos encargados del manejo de sesiones y la gestión de usuarios, sistemas y recursos; otra a los ficheros de configuración, en otra se encuentran las librerías a utilizar por los módulos anteriormente mencionados y una para los ficheros propios del framework que incluye el fichero que genera la base de datos correspondiente.

En la figura 18 puede verse la relación entre estos elementos físicos:

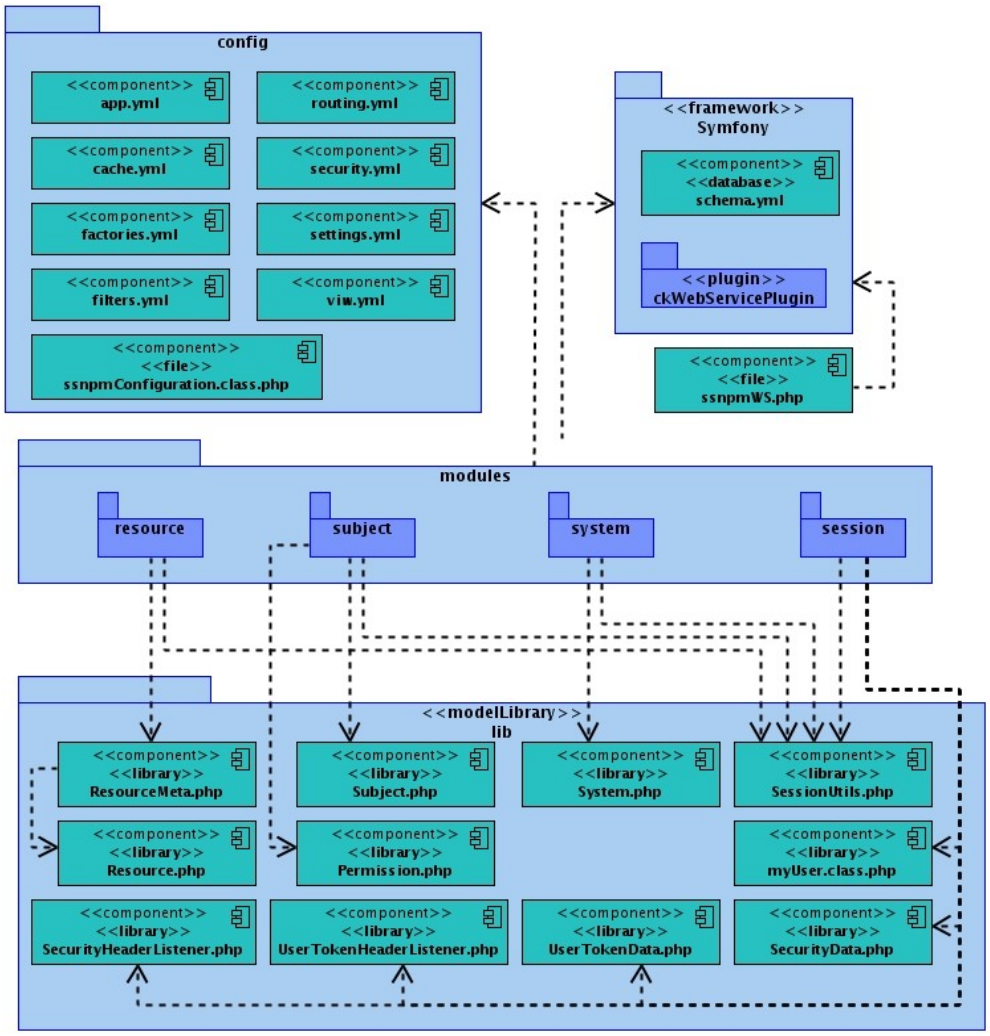

Figura 18: Diagrama de componentes del SCS

3.5 Modelo de datos

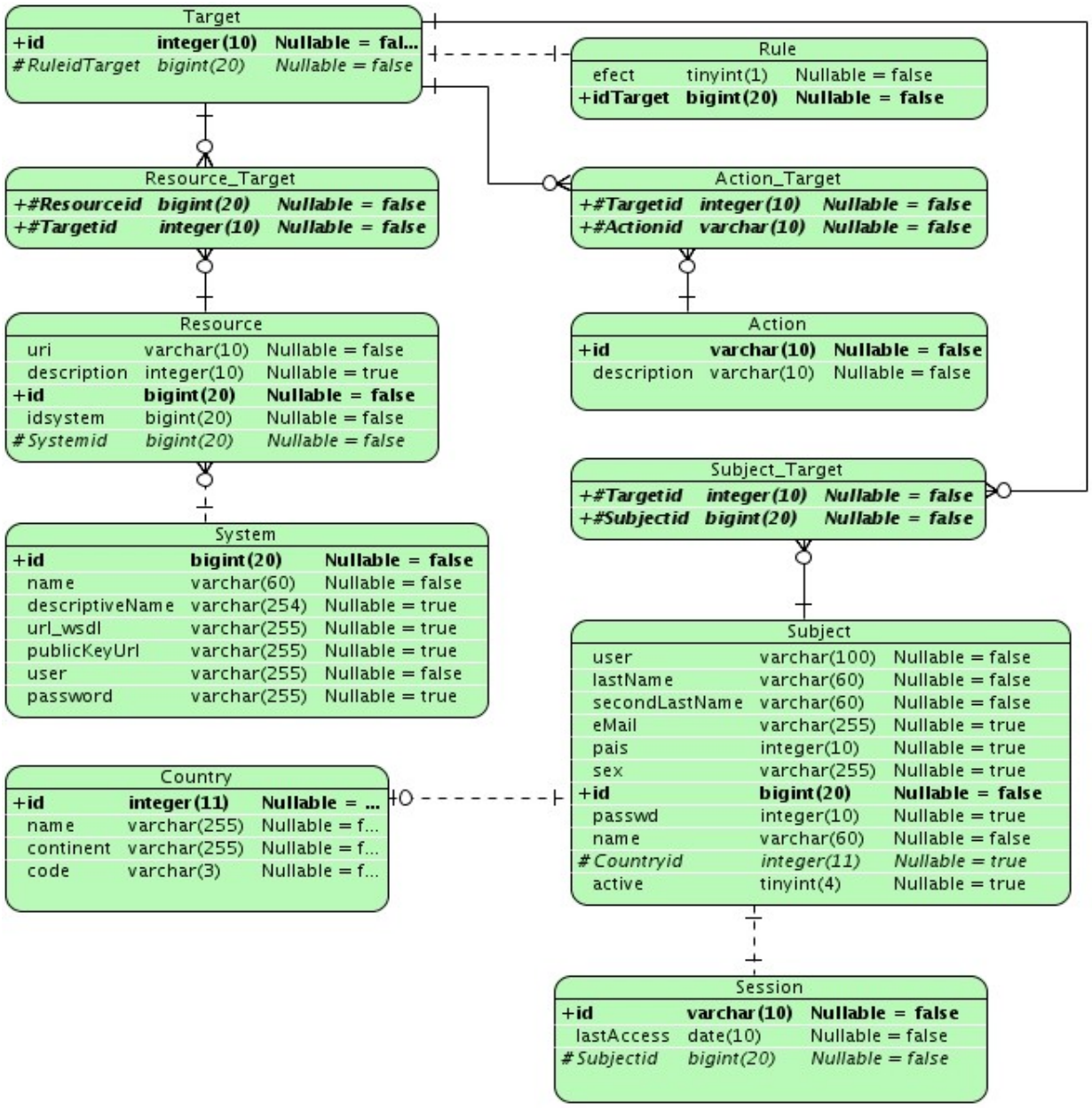

Figura 19: Diagrama del modelo relacional del SCS.

CAPÍTULO 4: SISTEMA DE MANEJO DE INTERFACES (SMI).

En el diseño de la arquitectura de la PMSWL, se utiliza, además de SOA, el estilo de presentación desacoplada, que implica una separación de la lógica para el manejo de la interacción de la representación de los datos con que trabaja el usuario. Entonces surge el SMI, el cual se debería encargar de gestionar las interfaces correspondientes al resto de los sistemas a integrar, a partir de sus respectivos MIU. Igualmente se definió que en este sistema estaría incluido un módulo que realizaría la gestión del proceso de migración.

El objetivo en este capítulo es realizar el diseño e implementación del SMI, a excepción del módulo anteriormente mencionado, el cuál será construido como resultado de otra investigación. Luego el resultado que se expone en este capítulo, puede verse como un pequeño framework que permitirá, una vez terminado, la implementación de los distintos MIU asociados a cada sistema.

4.1 Modelo de Diseño.

Este sistema sigue un diseño de clases con el uso de estereotipos web, en el que se reflejan tres páginas servidoras que se relacionan con una única página cliente, en el caso de *menú* y *contenido* lo hacen a través de llamadas AJAX, estas a su vez, incluyen funcionalidades contenidas en la clase *enrutador*, que puede considerarse como la controladora en el ámbito del sistema. La clase *enrutador* incluye varios ficheros PHP y además, se encuentra asociada directamente con *CoreContext*, *CoreMenu* y *Accion,* esta última hereda de la clase *CoreAccion* y está asociada también a *CoreServicio* y *CoreVista*.

CoreServicio es la clase que permite el acceso a los servicios web para ello provee automáticamente las credenciales necesarias, controla los errores que generen los servicios y establece un estándar para acceder a los resultados de los servicios web. CoreAccion provee una interfaz única para la ejecución de las diferentes acciones de los módulos. CoreContext tiene funcionalidades comúnmente usadas y que dependen del contexto. CorePaginar provee utilidades para el paginado de las vistas en las que se lista información.

La clase *Accion* se encuentra en el paquete *default,* que no es más que una representación genérica de cómo debe diseñarse cada MIU contenido en el SMI. Por tanto, se crearán tantas clases "*Accion*" como módulos tenga el sistema al que se le está construyendo el MIU, de igual forma cada una de estas clases incluirán tantos ficheros PHP como funcionalidades tenga el módulo al cual está asociado. Estos ficheros PHP se agrupan en paquetes del mismo nombre que tomen las clases "*Accion*" para garantizar un buen entendimiento. En la figura 20 se observa el diagrama de clases del diseño de este sistema y a continuación se explican las funcionalidades de las clases principales.

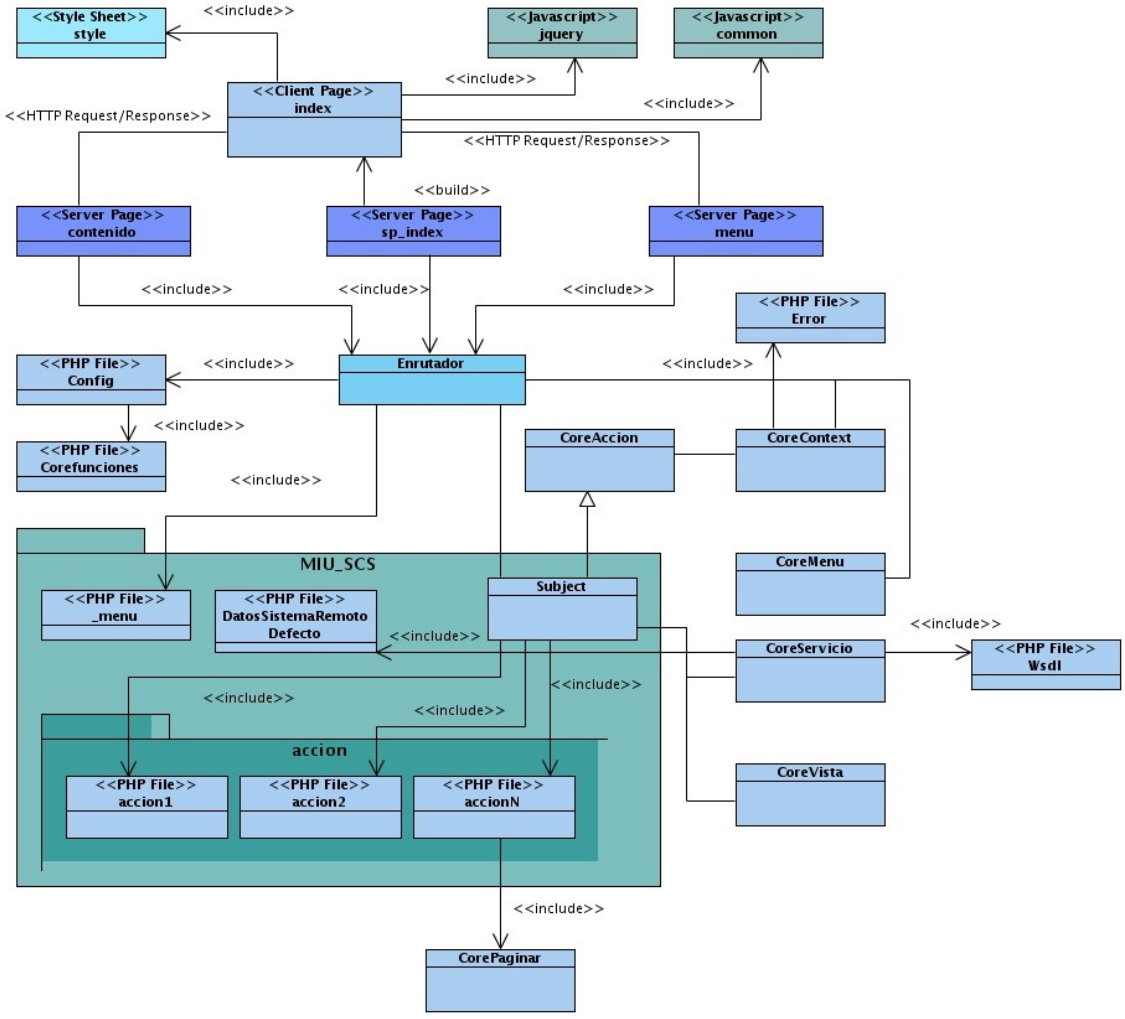

Figura 20: Diagrama de Clases del Diseño del SMI

4.1.1 Clase Enrutador.

Método	Descripción	Parámetros	Valor de Retorno
iniciar	Establece las configuraciones por defecto.	-	-
contenido	A partir de una petición ejecuta la acción solicitada.	-	-
isAjax	Se usa para verificar si una petición es AJAX o no.	-	bool (true si es AJAX)
menu	Carga el menú de un módulo dado.	-	-

4.1.2 Clase CoreContext.

Método	Descripción	Parámetros	Valor de Retorno
getParam	Retorna un parámetro pasado por GET o POST	$nombreparametro : string $requerido : bool (por defecto true)	Retorna el parámetro, de no existir retorna falso y si era requerido se adiciona el error a la cola.
existParam	Verifica la existencia de un parámetro.	$nombreparametro : string	bool
lanzarError	Adiciona un error	$mensaje : string $tipo : int 406 (por defecto) – Datos enviados no válidos 400 – Solicitud con sintaxis errónea. 401 – No autorizado 404 – Servicios web no	-

		encontrados	
acceso	Verifica si tiene permiso a usar un recurso de un sistema	$sistema : string $uri : string	bool (true si tiene permiso).

4.1.3 Clase CoreAccion.

Método	Descripción	Parámetros	Valor de Retorno
continuar	Permite saltar la ejecución de una acción a otra. Tiene un comportamiento similar a la función redirect aunque sólo se admiten redirecciones dentro del sistema.	$accion : string $modulo : string	-
ejecutar	Es la interfaz unificada para la ejecución de las acciones dentro de los módulos. Solamente es usado por la clase *Enrutador*.	$accion : string $grupoAccion : string $modulo : string	-

4.1.4 Clase CoreServicio.

Método	Descripción	Parámetros	Valor de Retorno
__construct	Se usa para llamar los servicios web. Si no se pasa usuario de sistema se verificará si existe un fichero (datosSistemaRemotoDefecto) en la configuración del módulo donde estén el usuario y la constraseña.	$idProveedorServicio : string $$tokenUsuario : string $usuarioSistema : string $passSistema : string	-
__call	Método mágico de PHP que se invoca ante la no existencia de un método en la clase.	$method : string $listaParams : array	-

4.1.5 Clase CoreFunciones.

Método	Descripción	Parámetros	Valor de Retorno
__	Encargado de la internacionalización, dado una cadena busca si tiene traducción al idioma en uso.	$key : string	string
logError	Registra errores.	$text : string $archivo : string $linea : int	-

4.2 Estructura de componentes.

Describir la estructura física de este sistema es esencial para garantizar el entendimiento de su funcionamiento, por el hecho de que los MIU que se construyan posteriormente deberán estar acoplados a la estructura que se defina, proporcionando uniformidad al sistema en general. La figura 21 muestra cómo quedó organizado el SMI, en términos de paquetes (que representan carpetas) y componentes.

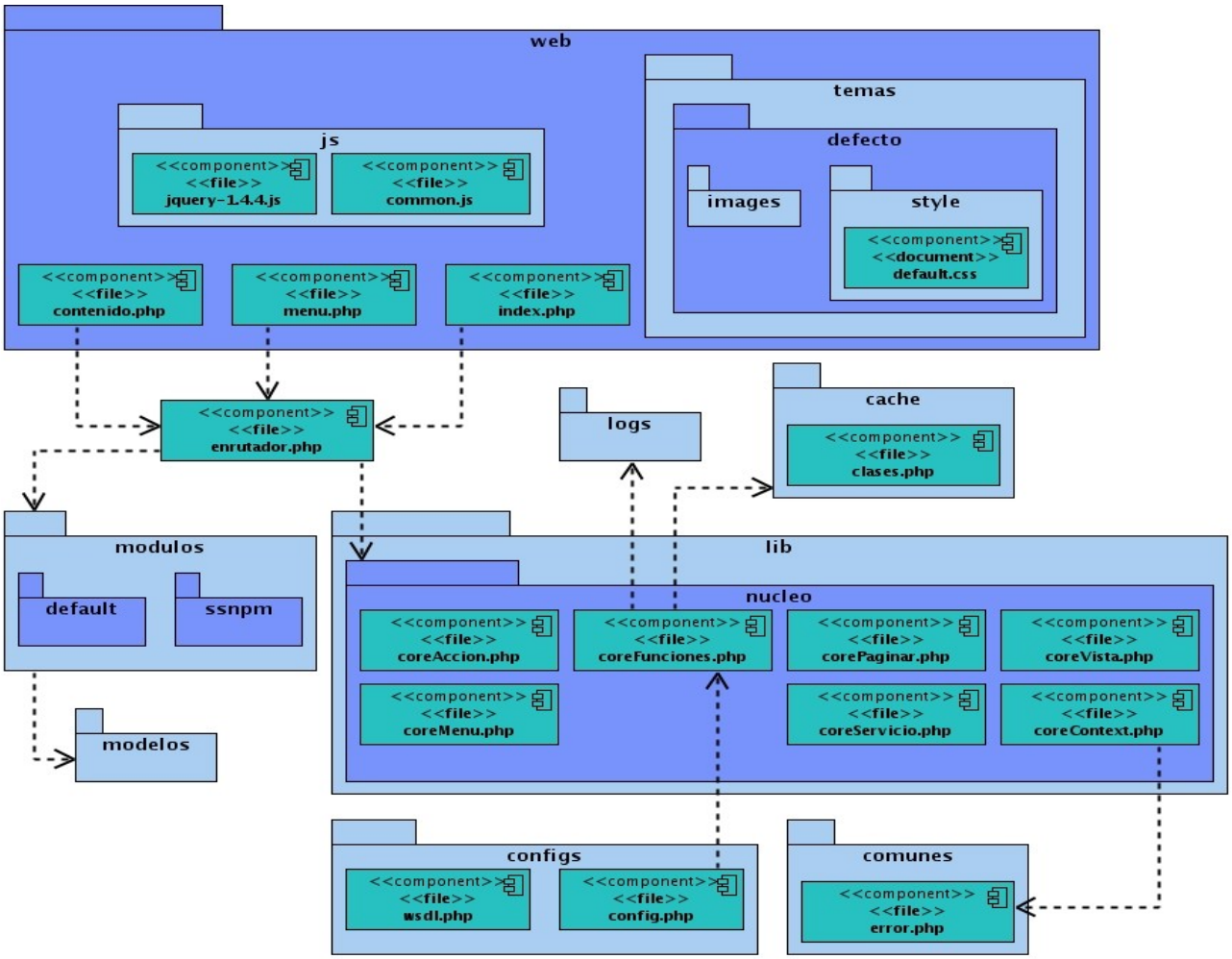

Figura 21: Diagrama de Componentes del SMI

La carpeta web abarca los ficheros javascript y los temas con sus respectivas imágenes y hojas de estilos, los que serán utilizados por la página cliente *index* mencionada en el epígrafe anterior. De igual forma están también incluidos los ficheros correspondientes a las tres páginas servidoras, las mismas dependen del fichero *enrutador.php* que representa a la clase del mismo nombre, este a su vez va a depender de los diferentes MIU que existan y estarán incluidos en la carpeta *módulos*, que por el momento sólo contiene las carpetas *default* y *ssnpm* que corresponde a la interfaz del sistema de seguridad que se explicará en el epígrafe siguiente.

La carpeta *web* y su contenido es la única que estará compartida usando el protocolo http, por razones de seguridad, no deberá ser posible el acceso directo desde la web al resto de las carpetas que conforman la estructura del SMI. La carpeta *lib*, contiene el código de las funcionalidades más importantes y que impactan a todo el sistema, no son ficheros que se modifiquen comúnmente.

4.3 Módulo de Interfaz de Usuario (MIU) del SCS.

Toda vez que se ha diseñado y construido el SMI, puede procederse al diseño e implementación de los MIU correspondientes a los sistemas de la PMSWL, con una estructura lógica similar a la que posee el paquete *default* explicado en el modelo de diseño del SMI. Teniendo en cuenta que se decidió utilizar el estilo presentación desacoplada en el diseño de la plataforma, los sistemas interactúan con sus respectivos MIU, por lo que este epígrafe se dedica a la construcción del MIU correspondiente al SCS referido en el capítulo anterior.

CAPÍTULO 5: DISCUSIÓN DE RESULTADOS.

Generalmente los procesos de desarrollo de software, en la mayoría de los casos, suelen ser caóticos y por ello se hace necesario involucrar procesos que permitan asegurar la calidad, para que de esta forma el producto desarrollado pueda cumplir con los requisitos que exige el cliente. El objetivo de este capítulo es explicar las pruebas realizadas a los componentes construidos y a la integración entre los sistemas de la PMSWL.

5.1 Casos de Prueba de Aceptación.

En la metodología SXP las pruebas se documentan en un artefacto denominado Caso de Prueba de Aceptación, en la que el desarrollador, el cliente y el probador comprueban y validan las funcionalidades del sistema a partir de las historias de usuario implementadas, para finalmente decidir la liberación del producto. En este epígrafe se muestran los Casos de Prueba de Aceptación más significativos.

Caso de Prueba de Aceptación	
Código Caso de Prueba: NPMSWL – 11 – 01	**Nombre Historia de Usuario:** Gestionar usuario
Nombre de la persona que realiza la prueba: Yoandy Pérez Villazón	
Descripción de la Prueba: Prueba a la funcionalidad adicionar un usuario a la plataforma, completando todos los campos correctamente.	
Condiciones de Ejecución: Haberse autenticado con una identidad que tenga permisos para gestionar usuarios. Acceder a la interfaz del SCS y seleccionar la opción adicionar un nuevo usuario.	
Entrada / Pasos de ejecución: 4 Completar los campos con los siguientes datos:	

• Usuario: avitier, Contraseña: 123456 (repetir para verificar coincidencia), Nombre: Abel, Apellido: García, Segundo Apellido: Vitier, Sexo: Masculino (seleccionar), Correo-e: avitier@estudiantes.uci.cu, País de Procedencia: Cuba (seleccionar), Activo: Sí (activar checkbox). 5 Click en el botón enviar.
Resultado Esperado: Debe aparecer la notificación de que los datos fueron insertados satisfactoriamente, el usuario agregado debe aparecer en la lista de usuarios (con los datos referidos), debe poderse autenticar en la plataforma utilizando el usuario y su respectiva contraseña.
Evaluación de la Prueba: Satisfactoria

Caso de Prueba de Aceptación	
Código Caso de Prueba: NPMSWL – 11 – 03	**Nombre Historia de Usuario:** Gestionar usuario
Nombre de la persona que realiza la prueba: Yoandy Pérez Villazón	
Descripción de la Prueba: Prueba a la funcionalidad editar un usuario en la plataforma.	
Condiciones de Ejecución: Haberse autenticado con una identidad que tenga permisos para gestionar usuarios. Acceder a la interfaz del SCS, al módulo de usuarios.	
Entrada / Pasos de ejecución: 1 Introducir “avitier” en el campo usuario del buscador. 2 Seleccionar el usuario y luego la opción editar. 3 Modificar los siguientes campos: • Contraseña: 12345678 (repetir para verificar coincidencia), Activo: No (desactivar checkbox). 4 Click en el botón enviar.	

Resultado Esperado: Debe aparecer la notificación de que los datos fueron insertados satisfactoriamente, el usuario agregado debe aparecer en la lista de usuarios como no activo, el sistema no debe permitir autenticarse en la plataforma utilizando el usuario y su respectiva contraseña.
Evaluación de la Prueba: Satisfactoria

Caso de Prueba de Aceptación	
Código Caso de Prueba: NPMSWL – 13 – 01	**Nombre Historia de Usuario:** Gestionar sistema
Nombre de la persona que realiza la prueba: Yoandy Pérez Villazón	
Descripción de la Prueba: Prueba a la funcionalidad adicionar un sistema en la plataforma.	
Condiciones de Ejecución: Haberse autenticado con una identidad que tenga permisos para gestionar sistemas. Acceder a la interfaz del SCS, al módulo de sistemas.	
Entrada / Pasos de ejecución: 1 Seleccionar la opción adicionar un nuevo sistema. 2 Completar los campos con los siguientes datos: • Nombre: directorio, Nombre descriptivo: Directorio de Software, Usuario: directorio, Contraseña: 12345678 (repetir para verificar coincidencia). 3 Click en el botón enviar.	
Resultado Esperado: Debe aparecer la notificación de que los datos fueron insertados satisfactoriamente, el sistema agregado debe aparecer en la lista de sistemas.	
Evaluación de la Prueba: Satisfactoria	

Caso de Prueba de Aceptación	
Código Caso de Prueba: NPMSWL – 13 – 02	**Nombre Historia de Usuario:** Gestionar sistema
Nombre de la persona que realiza la prueba: Yoandy Pérez Villazón	
Descripción de la Prueba: Prueba a la funcionalidad eliminar un sistema en la plataforma.	
Condiciones de Ejecución: Haberse autenticado con una identidad que tenga permisos para gestionar sistemas. Acceder a la interfaz del SCS, al módulo de sistemas.	
Entrada / Pasos de ejecución: 1 Buscar el sistema que se desea eliminar. 2 Seleccionar el sistema 3 Seleccionar la opción de eliminar.	
Resultado Esperado: Debe aparecer un mensaje de confirmación de eliminación y una vez aceptado el sistema debe mostrar un mensaje indicando que los datos fueron borrados. El sistema eliminado desaparece de la lista de sistemas.	
Evaluación de la Prueba: Satisfactoria	

Caso de Prueba de Aceptación	
Código Caso de Prueba: NPMSWL – 14 – 01	**Nombre Historia de Usuario:** Gestionar recurso de sistema
Nombre de la persona que realiza la prueba: Yoandy Pérez Villazón	
Descripción de la Prueba: Prueba a la funcionalidad adicionar un recurso de sistema.	

Condiciones de Ejecución: Haberse autenticado con una identidad que tenga permisos para gestionar recursos. Acceder a la interfaz del SCS, al módulo de recursos.
Entrada / Pasos de ejecución: 1 Seleccionar la opción de adicionar un nuevo recurso. 2 Completar los campos con los datos siguientes: • Uri: prueba1, Descripción: prueba, sistema: (seleccionar directorio) 3 Click en el botón enviar.
Resultado Esperado: Debe aparecer un mensaje indicando que los datos enviados correctamente. El nuevo recurso agregado debe aparecer en la lista de recursos correspondiente al sistema directorio.
Evaluación de la Prueba: Satisfactoria

Caso de Prueba de Aceptación	
Código Caso de Prueba: NPMSWL – 14 – 04	**Nombre Historia de Usuario:** Gestionar recurso de sistema
Nombre de la persona que realiza la prueba: Yoandy Pérez Villazón	
Descripción de la Prueba: Prueba a la funcionalidad mostrar recurso de sistema.	
Condiciones de Ejecución: Haberse autenticado con una identidad que tenga permisos para gestionar recursos. Acceder a la interfaz del SCS, al módulo de recursos.	
Entrada / Pasos de ejecución: 1 Seleccionar la opción de listar los recursos existentes. 2 En el filtro de búsqueda seleccionar el sistema ssnpm.	

3 Click en el botón aceptar.
Resultado Esperado: Debe mostrar todos los recursos asociados al sistema ssnpm, mostrando de cada uno sus respectiva uri y descripción. El máximo de recursos a mostrar se debe poder seleccionar en la parte inferior izquierda (3, 5 ó 10) apareciendo la cantidad de páginas en dependencia de la cifra seleccionada y la cantidad de recursos existentes.
Evaluación de la Prueba: Satisfactoria

Caso de Prueba de Aceptación	
Código Caso de Prueba: NPMSWL – 16 – 01	**Nombre Historia de Usuario:** Gestionar reglas
Nombre de la persona que realiza la prueba: Yoandy Pérez Villazón	
Descripción de la Prueba: Prueba a la funcionalidad adicionar nueva regla.	
Condiciones de Ejecución: Haberse autenticado con una identidad que tenga permisos para gestionar reglas. Acceder a la interfaz del SCS, al módulo de reglas.	
Entrada / Pasos de ejecución: 1 Seleccionar la opción de adicionar regla. 2 Completar los siguientes campos: Nombre: prueba5, Descripción: prueba 3 En la pestaña Usuarios, dar click en el botón Aceptar del filtro de usuarios para que muestre todos los usuarios disponibles. 4 Seleccionar cada usuario que se desee involucrar en la regla e ir agregándolo al campo Asignados a través del botón +. 5 En la pestaña Recursos proceder de la misma forma que con los usuarios para seleccionar los recursos a involucrar en la regla. Click en el botón enviar.	

Resultado Esperado: El sistema debe mostrar una notificación de que los datos fueron enviados correctamente. La regla insertada debe aparecer en la lista de reglas y los usuarios seleccionados deben tener acceso a los recursos que se asociaron en la regla creada.
Evaluación de la Prueba: Satisfactoria

Además de las pruebas realizadas a las funcionalidades definidas para el SCS y el SMI, se verificó además la integración de los sistemas en la PMSWL. Para esto se utilizaron los sistemas Directorio de Software y Homologación de Hardware por ser los más avanzados en el desarrollo. A través de la interfaz general se accedió a cada uno de los MIU correspondientes a estos sistemas, comprobándose que las funcionalidades responden satisfactoriamente. Otro aspecto esencial fue la prueba del funcionamiento de la seguridad en la PMSWL, para ello se eliminaron desde la interfaz del SCS los controles de acceso al Directorio de Software, comprobándose que automáticamente el mismo dejó de conceder acceso a sus servicios.

CONCLUSIONES

1. Para definir una estrategia de integración que reduzca el número de procesos y proporcione flexibilidad y capacidad de evolución al sistema, se debe diseñar una arquitectura determinando los estilos más apropiados y las tecnologías a utilizar.

2. Los requisitos técnicos y operacionales que permiten la integración de los sistemas componentes de la PMSWL se solucionan a partir de una arquitectura que combina los estilos SOA y Presentación Desacoplada y la construcción del SCS y el SMI.

3. Para llevar a cabo el manejo centralizado de la seguridad, se construye el SCS, que autoriza el acceso a los servicios, a partir de validaciones solicitadas por el resto de los sistemas.

4. La construcción de las interfaces de usuario de los sistemas de la PMSWL se garantiza a partir de un conjunto de funcionalidades que aporta el desarrollo del SMI.

5. A partir de las pruebas a las funcionalidades implementadas se comprobó que las mismas se encuentran listas para ser utilizadas y llevar a cabo la integración con los sistemas una vez que estos estén completamente funcionales.

RECOMENDACIONES

1. Desarrollar todos los sistemas definidos en la arquitectura de la PMSWL para que esta se encuentre completamente funcional, en especial el módulo de Gestión de Migración.

2. Validar el uso de la PMSWL con todas sus funcionalidades, en un proyecto de migración real.

3. Incluir en el SMI un diseñador gráfico que permita la construcción rápida y sencilla de interfaces de usuario.

REFERENCIAS BIBLIOGRÁFICAS

1. **Cala Hernández**. Propuesta de integración y nuevas herramientas para la Plataforma Cubana de Migración a Software Libre. 2010.
2. **Vargas**. Integración de Bases de Datos Mediante el uso de Portales Web. 2008. Available from world wide web: <http://www.trpconsultores.com/articles/Integracion%20de%20Bases%20de%20Datos%20con%20Portales.pdf>.
3. **Meier J.D., Hill David, Alex Hommer, and Jason Taylor**. Microsoft Application Architecture Guide, 2nd Edition. October 2009. [cited 10 November 2010]. Available from world wide web: <http://msdn.microsoft.com/en-us/library/dd673617.aspx>.
4. Idem 3.
5. **Roger Hill**. Layered Versus Client-Server. May 2007. [cited 12 November 2010]. Available from world wide web: <http://msdn.microsoft.com/en-us/library/bb421529.aspx>.
6. **Fowler Martin**. Separated Presentation. June 2006. [cited 15 November 2010]. Available from world wide web: <http://martinfowler.com/eaaDev/SeparatedPresentation.html>.
7. **Dahan Udi. Fear Those Tiers**. Diciembre 2007. [cited 11 November 2010]. Available from world wide web: <http://msdn.microsoft.com/en-us/library/cc168629.aspx>.
8. **Laribee David**. An Introduction To Domain-Driven Design. *MSDN Magazine* February 2009. [cited 10 November 2010]. Available from world wide web: <http://msdn.microsoft.com/en-us/magazine/dd419654.aspx>.
9. Ibidem 3.
10. **Endrei Mark et al.** *Patterns: Service-Oriented Architecture and Web Services*. 2004 RedBooks [cited 15 November 2010].
11. **Trowbridge, Roxburgh, Hohpe, and Manolescu**. Message Bus. June 2004. [cited 15 November 2010]. Available from world wide web: <http://msdn.microsoft.com/en-us/library/ms978583.aspx>.
12. **RAE**. Diccionario de la lengua española - Vigésima segunda edición. [cited 25 Octubre 2011]. Available from world wide web: <http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=tecnolog%EDa>.

13. **CollabNet Inc**. Sun OpenSSO Enterprise 8.0 Technical Overview - Sun Microsystems. [cited 21 November 2010]. Available from world wide web: <http://docs.sun.com/app/docs/doc/820-3740/adrab>.
14. **SimpleSAMphp Project**. SimpleSAMLphp. [cited 2 December 2010]. Available from world wide web: <http://simplesamlphp.org/>.
15. **Gonzalez Oyuela Sebastian**. JOSSO - Java Open Single Sign-On Project Home - Atricore. November 2010. [cited 2 December 2010]. Available from world wide web: <http://www.josso.org/confluence/display/JOSSO1/JOSSO+-+Java+Open+Single+Sign-On+Project+Home>.
16. **SpringSource**. Spring Security - Features. [cited 22 November 2010]. Available from world wide web: <http://static.springsource.org/spring-security/site/features.html>.
17. **W3C**. Guía Breve de Servicios Web. *W3C Oficina Española* 2010. [cited 10 December 2010]. Available from world wide web: <http://www.w3c.es/divulgacion/guiasbreves/ServiciosWeb>.
18. **W3C**. SOAP Specifications. [cited 14 December 2010]. Available from world wide web: <http://www.w3.org/TR/soap/>.
19. **Thomas Fielding**. Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). 2000. [cited 14 December 2010]. Available from world wide web: <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.
20. **Meneses Abad, Penalver Romero, Rodríguez Villar, Fernández Céspedes, and Pino García**. Metodología ágil para el desarrollo de proyecto de software libre. March 2009. [cited 4 January 2011]. Available from world wide web: <http://gforge.f10.uci.cu/plugins/scmsvn/viewcvs.php/*checkout*/Metodologia_SXP.odt?root=magmpr>.
21. **Penalver Romero**. Expediente para Metodología SXP. October 2008. [cited 20 January 2010]. Available from world wide web: <http://gforge.f10.uci.cu/plugins/scmsvn/viewcvs.php/*checkout*/SXP_con_CMMI_nivel_3/SXP_Expediente_de_Proyecto.odt?root=magmpr>.

BIBLIOGRAFÍA

Cala Hernández. Propuesta de integración y nuevas herramientas para la Plataforma Cubana de Migración a Software Libre. 2010.

Erika Camacho. Arquitecturas de software. Guía de Estudio. Abril 2004. [cited 11 Noviembre 2010]. Available from world wide web: <http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf>.

David Linthicum. Chapter 1: Service Oriented Architecture (SOA). 2004. [cited 7 Diciembre 2010]. Available from world wide web: <http://msdn.microsoft.com/en-us/library/bb833022.aspx>.

David Sprot, y Lawrence Wilkes. Understanding Service-Oriented Architecture. Enero 2004. [cited 7 Diciembre 2011]. Available from world wide web: <http://msdn.microsoft.com/en-us/library/aa480021.aspx>.

Iqbal Khan. MSDN Magazine: SOA Tips - Address Scalability Bottlenecks with Distributed Caching. Enero 2010. [cited 7 Diciembre 2011]. Available from world wide web: <http://msdn.microsoft.com/en-us/magazine/ff714590.aspx>.

OASIS Open. Service Oriented Architecture Reference Model. Octubre 2006. [cited 3 Noviembre 2010].

Paumier Samón, Pérez Villazón, Meneses Abad. Guía Cubana para la Migración a Software Libre. 2007. Available from world wide web: <https://repositorio.geitel.prod.uci.cu/svn/simays/resultados/guia-cubana-migracion-swl/documentos/serie-03/guia-cubana-0.32.pdf>.

Pérez Villazon. Metodología para la migración a software libre de las universidades del ministerio de educación superior. 2008. [cited 20 Octubre 2010]. Available from world wide web: <https://repositorio.geitel.prod.uci.cu/svn/simays/resultados/tesis/2007-2008/yoandy-perez-villazon/doc-final-tesis-yoandy-perez-villazon.pdf>.

GLOSARIO DE TÉRMINOS

Código Abierto: Es el término con el que se conoce al software distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código que a las cuestiones morales y/o filosóficas las cuales destacan en el llamado software libre.

Framework: Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Librería o Biblioteca: Es un conjunto de subprogramas utilizados para desarrollar software. Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de éstos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular. Algunos programas ejecutables pueden ser a la vez programas independientes y bibliotecas, pero la mayoría de éstas no son ejecutables. Ejecutables y bibliotecas hacen referencias (llamadas enlaces) entre sí a través de un proceso conocido como enlace, que por lo general es realizado por un software denominado enlazador.

Multiplataforma: Es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en Windows en un procesador x86, en GNU/Linux en un procesador x86, y en Mac OS X en uno x86 (solo para equipos Apple) o en un PowerPC.

Plataforma: Es un conjunto de artefactos (componentes o subsistemas) que forman una estructura común a partir de la cual se pueden derivar (desarrollar, construir) sistemas de una forma eficiente.

Los sistemas derivados de una plataforma no sólo comparten código, sino requisitos y arquitectura. Se da por tanto un proceso de reutilización natural de los artefactos.

SHA1: La familia SHA (Secure Hash Algorithm, Algoritmo de Hash Seguro) es un sistema de funciones hash criptográficas relacionadas con la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el National Institute of Standards and Technology (NIST). SHA-1 fue publicado en 1995 como sucesor de SHA-0.

Software libre: Es la denominación del software que respeta la libertad de los usuarios sobre su producto adquirido y, por tanto, una vez obtenido puede ser usado, copiado, estudiado, modificado y redistribuido libremente. Según la Free Software Fundation (FSF), el software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, modificar el software y distribuirlo modificado.

SXP: Es un híbrido de metodologías ágiles de desarrollo que toma las mejores prácticas de SCRUM y XP, además de regirse por los lineamientos de calidad de la UCI. Es iterativa e incremental, basada en Historias de Usuario (HU), está atenta al cambio y permite que el equipo de programación se mantenga en frecuente interacción con el cliente o usuario.

URI: Uniform Resource Identifier (en español "identificador uniforme de recurso") es una cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema.

WSDL: son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web. Describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.