Universidad de las Ciencias Informáticas

ARQUITECTURA DE INTEROPERABILIDAD DE APLICACIONES PARA LA UNIÓN DE INFORMÁTICOS DE CUBA

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

Autores
Tania Rodríguez Valera
Lester Collado Rolo

Tutor
Dra. Ailyn Febles Estrada

Co-tutor
Ing. Susana Cabeza Vidal
Ing. Roberto A. García Rodríguez
Ing. Michael Horta Fleitas

13 Junio de 2016
La Habana, Cuba

Declaración de autoría

Declaración de autoría

Declaramos que somos los únicos autores del trabajo "Arquitectura de interoperabilidad de aplicaciones para la Unión de Informáticos de Cuba" y autorizamos a la Facultad 4 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 _____ del año 2016.

Autor (es):

_____ _____

Tania Rodríguez Valera Lester Collado Rolo

Tutora:

_____

Dra. Ailyn Febles Estrada

"Cualquiera puede hacer complicado algo simple.

La creatividad consiste en hacer simple lo complicado"

Charles Mingus

Dedicatoria

De Lester:

A Fidel, por crear ideas grandiosas como la UCI.

A mis padres, por su infinito amor y dedicación.

A mi hermanita linda, Daílys.

De Tania:

A mi abuela Dora, que más que abuela fue mi segunda madre.

Agradecimientos

De Lester:

A la Revolución y a Fidel por darme la oportunidad de estudiar en esta casa de altos estudios y formarme como un profesional.

A mis padres Magdalena y Rubén, por ser guía constante de mi vida y estar incondicionalmente en cada momento de la vida, por el cariño, los años de sacrificio, por mi educación, por enseñarme a vivir, a luchar, a no rendirme, por todo, gracias.

A mi tata Daílys, por ser mi segunda mamá, por estar a mi lado apoyándome siempre y por ser el espejo en el cual mirarme para crecer como persona.

A mis sobrinitas, que son los tesoros de la casa, y a mi cuñado Yosnel por brindarme la mano de hermano.

A Yadelis por su paciencia, su compresión, su cariño y por ser siempre incondicional.

A mis suegros Félix y Massiel, por aceptarme como un miembro más de la familia.

A mis tíos que me han ayudado mucho durante todo el transcurso de mi vida.

A todos mis primos, en especial Yunior, Yunier y Manuel Alejandro, por siempre estar pendientes de mí.

A mi compañera de tesis Tania por soportarme durante todo este tiempo.

A mis tutores Ailyn, Roberto, Susana, Armando y a todas las personas que han aportado su granito de arena en este trabajo de diploma.

A mis amigos de la UCI: Oriesniel, David y Raydel por toda la amistad que me han transmitido durante los 5 años.

A otros amigos de la UCI que también aportaron mucho en los últimos tiempos: Carlos Montenegro, Antonio Gutiérrez (Tony), Yasirys Terry y Leonardo González.

A mis compañeros de apartamento y de grupo. Gracias por soportar mis jodederas y mis peleas durante tanto tiempo, en especial Rogelio, Franly, Henry, Jorge, Yaiselín, Jose, Andy, Roberto, Guillermo y Wendy.

Agradecimientos

A los amigos de la FEU por compartir tantos momentos de trabajo, alegrías y por sus enseñanzas.

A todos los profesores que me han acompañado y guiado en estos años en la universidad. Gracias a todos por compartir conmigo sus conocimientos y ayudarme ser una mejor persona. En especial Dunia María Colomé Cedeño, Maritza Calaña Hernández y Adrián García Sánchez.

Al consejo universitario y al consejo de dirección de la facultad por depositar en mi la confianza para cumplir con la enorme tarea de ser Presidente de la FEU.

A todos los que no mencioné y que me ayudaron durante mis estudios. Son muchos durante estos 5 años. A todos los que me preguntaban siempre: ¿y la tesis cómo va?, ¿cuándo te gradúas?, a todos los que hicieron este sueño realidad.

A todos ellos, muchas gracias.

Agradecimientos

De Tania:

A mi mamá, por su apoyo y sobre todo por su amor incondicional. Porque ha estado presente en todos los momentos de mi vida, en los buenos y sobre todo en los malos.

A Yelko, por ser el mejor padre que pudiera desear. Porque a pesar de mis majaderías siempre ha estado ahí cuando lo he necesitado.

A mi abuelo, porque me ha apoyado siempre en todos mis sueños y me ha motivado a ser siempre una mejor persona.

A mis tíos, que de una forma u otra se han preocupado siempre por mí.

A mi compañero de tesis, por aguantar todo mi estrés.

A mis tutores y a todas las personas que de una forma u otra nos han apoyado.

A mis amigos de los años: Jorge Eduardo, Giselle, Carlos, Jessica, Javier, Cynthia, Jerson, Wenkin; por todos los años y todos los momentos que hemos compartido juntos. Por soportarme con todos mis defectos y mi mal humor.

A mis compañeras de apartamento: Wendys, Yaiselín y Edelín, por su amistad y por apoyarme en las buenas y en las malas.

A Jorgito, Henrry y Rogelio, que han sido mis compañeros de fiestas, playas y juegos de dominó; porque siempre me han hecho reír y me han cuidado como a una hermana en todo momento.

A Takeshi, Hugo, Luciano, Arletys, Rachelys, Jose y todas las personas que durante estos 5 años he llegado a considerar mis amigos.

A todos los profesores que me han ayudado durante estos 5 años y a todas las personas que quizás no mencioné y que de una forma u otra me han ayudado durante mis estudios.

Muchas gracias.

Resumen

La Unión de Informáticos de Cuba surge con el objetivo de crear un espacio para el intercambio de experiencias y conocimientos por parte de los profesionales de la rama de la informática y carreras afines a la misma. Para llevar a cabo sus procesos de manera eficiente, la organización necesita emplear servicios que ofrecen diferentes sistemas informáticos que se emplean en la Universidad de las Ciencias Informáticas, sin embargo por sus características, los mismos no pueden interactuar unos con otros. Ante esta situación, fue necesario diseñar una arquitectura de interoperabilidad que posibilitara dicha integración, respondiendo a atributos como funcionalidad, modificabilidad, portabilidad, interoperabilidad, etc. Entre los aspectos que influyeron para alcanzar estos atributos de calidad se encuentra el estudio de algunas de las soluciones de interoperabilidad existentes a nivel nacional e internacional, lo cual permitió identificar a la arquitectura orientada a servicios como la más adecuada para desarrollar la propuesta de solución. Como resultado de este análisis fue posible también la definición de los estándares y herramientas que forman parte de la arquitectura de interoperabilidad y posteriormente la evaluación de la misma a través de métodos que permitieron evaluar los atributos de calidad deseados en los escenarios definidos. A partir de esta evaluación se comprobó que la arquitectura cumple con los requerimientos planteados y satisface los atributos que se desean alcanzar permitiendo lograr el objetivo propuesto.

Palabras clave: arquitectura de interoperabilidad, atributos de calidad, estándares.

Índice

Introducción

A nivel mundial son múltiples las organizaciones encargadas de agrupar a un número considerable de personas de acuerdo a sus intereses, sus competencias o a la rama del conocimiento a la que pertenecen. Las mismas generalmente fomentan la cooperación y el trabajo en equipo como una alternativa más simple y eficiente para el cumplimiento de sus objetivos o tareas. Cuba no está ajena a este proceso y cuenta en la actualidad con varias instituciones de este tipo, creadas con el afán de aunar los esfuerzos de muchos individuos aislados para lograr mejores resultados.

La creciente formación de profesionales en la rama de la informática y carreras afines a la misma a lo largo del país y el empleo masivo de las Tecnologías de la Información y las Comunicaciones (TIC), ha generado la necesidad de crear un espacio para el intercambio de experiencias y conocimientos por parte de dichos profesionales. Como respuesta a esta necesidad surge la Unión de Informáticos de Cuba (UIC). Como se plantea en los artículos 1 y 4 de sus estatutos, se define la misma como: *"…una organización social de profesionales de las Tecnologías de la Información y las Comunicaciones (TIC) sin ánimo de lucro autofinanciada, con un perfil científico-profesional que se constituye al amparo de lo dispuesto en el artículo 7 de la Constitución de la República de Cuba, se basa en la más firme unidad de todos sus miembros en torno al proyecto social y el modelo económico impulsado por la Revolución cubana, así como a su esfuerzo transformador.", "…tiene carácter social y alcance nacional con estructuras territoriales en cada una de las provincias y el municipio Especial Isla de la Juventud, su sede principal radica en La Habana."*(1)

La organización, única en su tipo en el país, tiene entre sus objetivos ser un referente del sector de las tecnologías para Cuba y representar a sus profesionales en el extranjero. Así como lograr el intercambio de experiencias y conocimientos, propiciar la creación científico-técnica de forma que motive a sus afiliados a alcanzar la superación profesional y contribuir desde esta área al desarrollo social del país (1).

La UIC cuenta con un portal institucional que se encuentra en desarrollo para ser convertido en un sitio web dinámico, capaz de mostrar los contenidos de acuerdo a las exigencias del usuario. Actualmente, para el manejo de la gestión documental, la institución emplea el sistema eXcriba, implementado en la Universidad de las Ciencias Informáticas (UCI). A partir del mismo se gestionan los datos referentes a sus miembros y se generan reportes relativos al proceso de inscripción y aprobación de sus afiliados. Tomando estos datos como base, la organización necesita ser capaz de

extraer la información necesaria relativa a los miembros para ser transferidos a un sistema de identificación.

La institución cuenta con un presupuesto centralizado y necesita realizar la gestión contable apoyándose en el uso de un ERP[1]. Para ello es necesario definir una estructura de este sistema que al integrarse en una plataforma, permita controlar el pago de la cotización por provincia, por persona y por delegación de base. A su vez, por las características del presupuesto, a este sistema solo deben tener acceso los miembros vinculados a la sede central.

Por otra parte, es necesario incorporar a esa plataforma un sistema de planificación de actividades (SIPAC), con acceso nacional, en el que se publiquen los planes provinciales. Derivado de este sistema, debe ofrecerse un servicio de publicación de las actividades a realizarse mensualmente, planificadas por el SIPAC. Actualmente, estos sistemas operan de forma independiente en la UCI por lo que solo pueden ser utilizados por usuarios internos de la universidad, lo que impide un acceso a los servicios desde las diferentes provincias del país. Además, no existe una plataforma que los integre y le permita a la UIC hacer uso de los servicios que ellos ofrecen.

Tomando como punto de partida de esta investigación la situación descrita previamente, se identifica el siguiente **problema científico**: ¿cómo lograr la interoperabilidad entre aplicaciones que brindan servicios necesarios para la UIC?

De lo planteado anteriormente se deriva como **objeto de estudio:** las arquitecturas de software, y como **campo de acción**: la arquitectura que permita la interoperabilidad entre aplicaciones.

Para dar solución al problema planteado se define como **objetivo general:** diseñar una arquitectura que permita la interoperabilidad entre aplicaciones que brindan servicios necesarios para la UIC.

De este se derivan los siguientes **objetivos específicos:**

- Realizar un estudio del estado del arte relacionado con el campo de acción.
- Realizar un estudio de los estándares y herramientas que se emplean en el desarrollo de la propuesta de solución.

[1] Enterprise Resource Planning

- Proponer una arquitectura que permita la interoperabilidad entre aplicaciones.
- Validar la propuesta de solución.

Teniendo como **preguntas científicas:**

- ¿Cómo lograr un punto único de acceso a la información que maneja la UIC?
- ¿Qué estándares y herramientas deben emplearse para lograr la interoperabilidad entre aplicaciones heterogéneas?
- ¿Qué características debe poseer la arquitectura para permitir la interoperabilidad entre aplicaciones?

El **posible resultado** de esta investigación sería**:** una arquitectura que garantice la interoperabilidad de las aplicaciones que brindan servicios necesarios para la UIC.

Para dar cumplimiento a los objetivos específicos se proponen las siguientes **tareas de investigación:**

- Realización de un estudio para la elaboración de un marco teórico-conceptual sobre arquitecturas que permitan la interoperabilidad entre aplicaciones.
- Diseño de la arquitectura que permita la interoperabilidad entre aplicaciones que brindan servicios necesarios para el funcionamiento de la UIC.
- Definición de los instrumentos y métodos para la validación de la propuesta de solución.
- Validación de la arquitectura propuesta.

Los **métodos científicos** utilizados estuvieron determinados por el objetivo general y las tareas de investigación previstas. Los mismos son:

A nivel empírico:

Observación: posibilitó la toma de las prácticas positivas de sistemas similares con el mismo fin de la propuesta de solución, así como la identificación de vulnerabilidades comunes en la puesta en funcionamiento de las mismas para convertirlas en ventajas durante el desarrollo de la investigación.

A nivel teórico:

Análisis-síntesis: fue utilizado durante el proceso investigativo para realizar un análisis de la documentación relacionada con las arquitecturas de software, y los conceptos vinculados con la

interoperabilidad de aplicaciones. Facilitó el análisis de las características de las herramientas y estándares empleados. Esto permitió realizar posteriormente la síntesis de los aspectos más importantes para el desarrollo del trabajo.

Análisis histórico-lógico: permitió realizar un análisis sobre las tendencias en el empleo de arquitecturas de software para permitir la interoperabilidad entre aplicaciones. Además, sirvió para identificar soluciones similares que aportaron elementos importantes para la solución propuesta.

Modelación: permitió representar los conceptos vinculados con la propuesta de solución y las relaciones entre ellos para lograr un mejor entendimiento de los mismos.

Para una mejor comprensión de la presente investigación, se definió una **estructura capitular** que facilite el estudio del documento y provea organización al mismo.

En el **Capítulo 1: Fundamentación teórica,** se expresan los conceptos relacionados con el tema de investigación y las definiciones asumidas por los autores. Se definen los patrones de integración a emplear durante el desarrollo de la propuesta de solución y se realiza un estudio del estado del arte de las arquitecturas que permiten la interoperabilidad entre aplicaciones.

En el **Capítulo 2: Descripción de la propuesta de solución,** se analizan los estándares y herramientas más usados para el desarrollo de arquitecturas de software y de los mismos se seleccionan los más adecuados para emplearlos en la propuesta de solución. Se describe la arquitectura propuesta, especificando la función de cada una de sus capas y cómo se realiza la interacción entre las mismas.

En el **Capítulo 3: Validación de la arquitectura propuesta,** se definen conceptos esenciales vinculados a la validación de arquitecturas, se describen distintos métodos de evaluación y se justifica la selección del método de Losavio para la validación de la arquitectura propuesta. Se ofrecen además, los resultados de la aplicación del método.

Capítulo 1: Fundamentación teórica

El objetivo de este capítulo es profundizar en los conceptos esenciales relacionados con las arquitecturas de software que permiten la interoperabilidad entre aplicaciones. Se analizan los patrones que permiten la integración, además de realizar un estudio de algunos sistemas similares vinculados al campo de acción.

1.1 Conceptos generales relacionados con la investigación

Interoperabilidad

En la actualidad, es común que las empresas no cuenten con un único sistema informático capaz de dar solución a todos los procesos que estas llevan a cabo, sino más bien que en su infraestructura tecnológica, debido a muchos factores, exista una variedad de estas, respondiendo a diversas plataformas de desarrollo, distintos lenguajes y arquitecturas. Esta situación da lugar a la insuficiente comunicación entre estos sistemas informáticos, provocando la fragmentación de los procesos del negocio en las empresas. La información estratégica de la entidad queda dividida por áreas, departamentos o direcciones y no se tiene una colaboración mutua y coordinación entre las actividades o procesos que se pudieran llevar a cabo con más agilidad si los sistemas se entendieran entre sí (2). Lograr la interoperabilidad entre dichos sistemas ofrece una solución a este problema.

Según el EIF[2] la interoperabilidad es *"...la habilidad de los sistemas TIC, y de los procesos de negocios que ellas soportan, de intercambiar datos y posibilitar el compartimiento de información y conocimientos".* Por su parte, Ruiz Palacios (3) la define como: *"la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada."*

A partir del estudio de diferentes bibliografías referentes al tema y tomando como base las definiciones anteriores, los autores del presente trabajo de diploma asumen como interoperabilidad: *la capacidad de los sistemas para interactuar, intercambiar información y ejecutar tareas de manera*

[2] European Interoperability Framework

conjunta mediante el empleo de mecanismos que permitan realizar estos procesos entre sistemas heterogéneos.

Existen disímiles tipos de interoperabilidad, que pueden ser definidos basándose en dos puntos de vista, el de contenidos y el de la organización. Para el desarrollo de la presente investigación se optará por este último, el cual define los siguientes tipos de interoperabilidad:

Interoperabilidad técnica: garantiza la compatibilidad entre sistemas, interfaces, formatos y protocolos. La misma se alcanza con el empleo de especificaciones y estándares abiertos (4).

Interoperabilidad semántica: consiste en el empleo de tecnologías específicas que permitan el intercambio de información, mediante la utilización de un lenguaje común para el manejo de los datos (5).

Interoperabilidad organizativa: se refiere a la definición de objetivos de la organización, a los procesos de gestión y al establecimiento de medios para colaborar con otras administraciones en el intercambio de información. Garantiza el incremento de la productividad, la simplificación de las estructuras organizativas y la reducción de costos (5).

Arquitectura de Software

A pesar de que el término arquitectura de software (AS) se viene empleando por más de cuatro décadas, no es hasta 1992, con la publicación de un estudio de Perry y Wolf, cuando se hace referencia a esta definición en el sentido en que actualmente es conocido. El número de definiciones existentes es amplio, sin embargo, se ha acordado que la definición oficial de AS sea la que brinda el documento de IEEE[3] Std 1471-2000, adoptada también por Microsoft, el cual plantea lo siguiente:

"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución." Bass, Clements y Kazman (6) la definen como: *"…la estructura o estructuras del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos."*

[3] Institute of Electrical and Electronics Engineers

Los autores del presente trabajo, se acogen sin embargo, a la definición ofrecida por Roger Pressman en la séptima edición de su libro “Ingeniería del software. Un enfoque práctico”, donde se hace referencia a la misma como una representación que permite:

- Analizar la efectividad del diseño para cumplir los requerimientos establecidos.
- Considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil.
- Reducir los riesgos asociados con la construcción del software.

La importancia de la arquitectura de software radica fundamentalmente en que: resalta las primeras decisiones, las cuales tendrán un gran impacto en todo el trabajo de ingeniería de software, así como en el éxito del sistema como una entidad operacional; sus representaciones permiten la comunicación entre las partes interesadas en el desarrollo de un sistema, y constituye un modelo relativamente pequeño y asequible sobre cómo está estructurado el sistema y la forma en que sus componentes trabajan juntos (7).

Arquitectura de interoperabilidad

Muchas personas tienden a pensar que una arquitectura de interoperabilidad es lo mismo que una arquitectura de software. En el libro blanco de interoperabilidad Moreno Escobar, et al. (8), plantean que la misma ayuda a definir la forma en que las aplicaciones serán construidas, cómo se desarrollarán los componentes y servicios, y sobre todo, cómo podrán interactuar con los demás sistemas con los que se cuenta. Cuando se define una arquitectura de interoperabilidad, se estandarizan los procesos, y se define la estructura de los datos que se intercambian.

En general, la arquitectura de interoperabilidad corresponde al *conjunto de estándares y directrices que describen la forma en la cual las organizaciones han establecido, o pueden establecer, los mecanismos para interactuar unas con otras, teniendo en cuenta elementos o aspectos como la seguridad, autenticidad e integridad de los datos que son objeto de esta interacción* (9).

1.2 Patrones de integración

Aunque la integración es un tema muy amplio y difícil, se han resuelto bastantes problemas definiendo diseños comunes en el desarrollo de funcionalidades relacionadas con la integración de aplicaciones, esto se conoce como patrones de integración. Dichos patrones especifican una manera

estándar de realizar ciertas tareas y ayudan a conocer con un lenguaje común determinados objetos que se desarrollan habitualmente. Con el decursar de los años han evolucionado por ensayo y error o de otros arquitectos de integración con experiencia. A continuación, se realiza un estudio de los patrones de integración, tomando como herramienta algunos de los escenarios de integración más comunes: (10)

Portal de Información

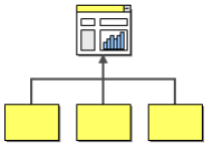

Figura 1. Patrón de integración "Portal de Información". (Fuente: [Hohpe, 2003])

Muchos usuarios de negocio tienen que acceder a más de un sistema para responder a una pregunta específica o para realizar una sola función empresarial. Por ejemplo, para verificar el estado de un pedido, un representante de servicio al cliente puede necesitar tener acceso al sistema de gestión de pedidos en el ordenador central demás de iniciar sesión en el sistema que gestiona los pedidos realizados a través de Internet.

EI patrón Portal de Información muestra información agregada de múltiples fuentes en una sola pantalla para evitar los accesos innecesarios a múltiples sistemas informáticos. En su implementación más simple divide la pantalla en varias zonas, cada una de las cuales muestra la información de un sistema diferente. En versiones más sofisticadas proporciona una interacción limitada entre zonas, por ejemplo, cuando un usuario selecciona un elemento de una lista en la zona A, la zona B se actualiza con información detallada sobre el elemento seleccionado.

Replicación de datos

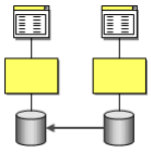

Figura 2. Patrón de integración “Replicación de datos”. (Fuente: [Hohpe, 2003])

Los diferentes sistemas de negocios necesitan tener acceso a los mismos datos. Por ejemplo, la dirección de un cliente puede ser utilizada en un sistema de atención al cliente cuando se desea hacer un cambio en un pedido, en un sistema de contabilidad para calcular el impuesto de ventas, en

un sistema de envío para etiquetar el envío y en un sistema de facturación para enviar una factura. Muchos de estos sistemas van a tener sus propios almacenes de datos para guardar la información relacionada con el cliente. Cuando un cliente llama para cambiar su dirección entonces todos estos sistemas tienen que cambiar su copia del dato. Esto se puede lograr mediante la aplicación de una estrategia de integración basada en la replicación de datos. Hay muchas maneras diferentes de implementar la replicación de datos. Por ejemplo, algunos proveedores de bases de datos acumulan funciones de replicación que permiten exportar datos en archivos y volver a importarlos en otro sistema, o puede usarse un *middleware* orientado a mensajes para el transporte de registros de datos dentro de los mensajes.

Función de Negocio Compartida

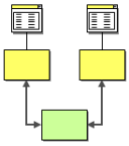

Figura 3. Patrón de integración "Función de negocio compartida". (Fuente: [Hohpe, 2003])

De la misma manera que muchas aplicaciones de negocios almacenan datos redundantes, tienden también a implementar funcionalidades redundantes. Múltiples sistemas pueden necesitar la misma funcionalidad, por tanto, tiene sentido exponer las funciones una vez implementadas y ofrecerlas como servicios a otros sistemas.

Una función de negocio compartido puede abordar algunas de las mismas necesidades que la replicación de datos. Por ejemplo, se podría implementar una función denominada "Obtener dirección del Cliente", que podría permitir a otros sistemas solicitar la dirección del cliente cuando es necesario y no siempre guardar una copia redundante. La decisión entre estos dos enfoques es impulsado por una serie de criterios, tales como la cantidad de control que se tiene sobre los sistemas (llamar a una función compartida suele ser más intrusivo que cargar datos en la base de datos) o la tasa de cambio (una dirección puede necesitar cambiar con frecuencia, pero con muy poca frecuencia).

Arquitectura Orientada a Servicios

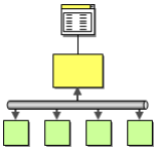

Figura 4. Patrón de integración "Arquitectura Orientada a Servicios". (Fuente: [Hohpe, 2003])

Este patrón está basado en los servicios que se implementan para compartir funciones de negocios. Un servicio es una función bien definida que es universalmente disponible y responde a las peticiones de "consumidores de servicios". Una vez que una empresa reúne una colección de servicios útiles, la gestión de los servicios se convierte en una función importante. En primer lugar, las aplicaciones necesitan alguna forma de directorio de servicios, o sea, una lista centralizada de todos los servicios disponibles. En segundo lugar, cada servicio necesita describir su interfaz de tal manera que una aplicación pueda "negociar" un contrato de comunicaciones con el servicio. Estas dos funciones: el descubrimiento de servicios y la negociación, son los elementos clave que conforman una arquitectura orientada a servicios.

Las Arquitecturas Orientadas a Servicios (SOA) desdibujan la línea entre la integración y las aplicaciones distribuidas. Una nueva aplicación se puede desarrollar utilizando los servicios existentes que a su vez pueden ser proporcionados por otras aplicaciones remotas. Por lo tanto, llamar a un servicio se puede considerar la integración entre las dos aplicaciones. Por otro lado, una arquitectura orientada a servicios por lo general proporciona herramientas que hacen que la petición de un servicio externo sea casi tan simple como llamar a un método local.

Proceso de Negocio Distribuido

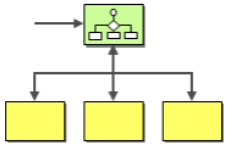

Figura 5. Patrón de integración "Proceso de negocio distribuido". (Fuente: [Hohpe, 2003])

Uno de los principales impulsores de la integración es el hecho de que una sola transacción de negocios a menudo se propaga a través de muchos sistemas diferentes. En la mayoría de los casos, todas las funciones relevantes se incorporan dentro de las aplicaciones existentes, lo que falta es la coordinación entre las mismas. Este patrón añade un componente de gestión de procesos de negocio que maneja la ejecución de una función de negocio a través de múltiples sistemas existentes. Los límites entre una arquitectura orientada a servicios y una empresa distribuida pueden desdibujarse. Por ejemplo, se podrían exponer todas las funciones empresariales relevantes como el servicio y luego codificar los procesos de negocio dentro de una aplicación que tiene acceso a todos los servicios a través de una SOA.

Integración Negocio a Negocio

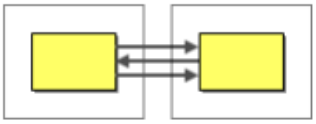

Figura 6. Patrón de integración "Integración Negocio a Negocio". (Fuente: [Hohpe, 2003])

Los patrones anteriores han considerado principalmente la interacción entre las aplicaciones y funciones de negocios dentro de una empresa. En muchos casos, las funciones de negocio pueden estar disponibles para proveedores externos o socios comerciales. Por ejemplo, una compañía naviera puede proporcionar un servicio para los clientes que permita calcular los costos de envío o de pista, o una empresa puede utilizar un proveedor externo para calcular las tasas de impuestos sobre las ventas. Las consideraciones anteriores se aplican igualmente a la integración negocio a negocio. Sin embargo, la comunicación a través de Internet o cualquier otra red plantea nuevas cuestiones relacionadas con los protocolos de transporte y seguridad.

Teniendo en cuenta el ambiente tecnológico de la Unión de Informáticos de Cuba, se determina utilizar en esta investigación el patrón de integración SOA; permitiendo a través de los componentes de una Arquitectura Orientada a Servicios que se comuniquen las aplicaciones informáticas que necesita la entidad y otras que puedan desarrollarse.

1.3 Arquitectura Orientada a Servicios

SOA comienza a tener auge en el mercado en el año 2000 y desde entonces son múltiples las definiciones que se han dado sobre este modelo. La W3C[4] la define como un *"conjunto de componentes que pueden ser invocados, cuyas descripciones de interfaces se pueden publicar y descubrir"* (13), el CBDI[5] plantea que es un *"estilo resultante de políticas, prácticas y frameworks que permiten que la funcionalidad de una aplicación se pueda proveer y consumir como conjuntos de servicios, con una granularidad relevante para el consumidor…"* (13)

[4] World Wide Web Consortium
[5] Centro Brasileño de Desarrollo en Informática

De manera general, la misma representa una metodología para lograr interoperabilidad entre aplicaciones y servicios web de manera tal que permita reutilizar tecnologías de información ya existentes. Se constituye de servicios que se exponen y servicios que se consumen; difiriendo del tradicional enfoque cliente/servidor y haciendo énfasis en el bajo acoplamiento entre los componentes de software. Permite documentar las competencias del negocio, el enlazado de dependencias en tiempo de ejecución, la reutilización de servicios y el recambio por otros de funcionalidad similar. Aunque puede ser implementada a través de otras tecnologías, lo más recomendable es el empleo de servicios, debido a que los mismos brindan soporte a los requisitos del software, permitiendo de forma ágil, la creación y modificación de los procesos del negocio desde la perspectiva de las TI (14).

Entre sus ventajas, pueden contarse, un menor costo de integración, mayor flexibilidad, reutilización de los activos y servicios existentes, y un enfoque basado en estándares e interoperabilidad. Las organizaciones que implementan satisfactoriamente esta arquitectura, al tener sus servicios alineados con los negocios estratégicos de TI, pueden reaccionar más rápido a los cambios en los requerimientos de negocio que las que no los tienen (15).

El principal objetivo de SOA es lograr un alto desempeño desarrollando servicios que puedan ser reutilizados por la organización. Estas capacidades permiten que las tecnologías respondan rápidamente a los cambios de los negocios, los escenarios competitivos, las alianzas y las necesidades de los consumidores. SOA permite la creación de aplicaciones compuestas que trabajan a través de una única interfaz, facilitando la comprensión de distintos procesos de negocios (16). Este modelo presenta grandes ventajas en su adopción por parte de las empresas, entre las mismas pueden contarse:

- Mejora en los tiempos de realización de cambios en procesos.
- Facilidad para evolucionar a modelos de negocios basados en tercerización.
- Facilidad para abordar modelos de negocios basados en colaboración con otros entes (socios, proveedores).
- Poder para reemplazar elementos de la capa aplicativa SOA sin generar interrupciones en el proceso de negocio.
- Proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

De forma general las características principales de SOA son (17):

- Los servicios deben estar definidos a través de protocolos y lenguajes estándar de forma que su uso sea independiente de la plataforma.
- Los servicios deben estar accesibles y publicados en un repositorio con el objetivo de monitorizar su estado.
- Los servicios definidos deben cumplir el requisito de reutilización. Se debe fomentar el uso de estándares para reutilizar servicios ya existentes con el objetivo de aumentar las posibilidades de integración con sistemas legados.
- Los servicios encapsulan los detalles de implementación y exponen sus funcionalidades a través de una interfaz basada en estándares.
- Un servicio debe ser independiente del estado de otro, por tanto sus relaciones deben ser débilmente acopladas.
- Los servicios pueden reorganizarse en una orquestación de servicios para representar funcionalidades más complejas o integraciones de los procesos de negocio.

La arquitectura orientada a servicios contiene tres actores (18):

- Un solicitante de servicio: es responsable de encontrar una descripción de servicio publicada en uno o más registros, y de utilizar las mismas para invocar los servicios web hospedados por los proveedores de servicios.
- Un proveedor de servicio: es responsable de crear una descripción de servicio, publicando la misma en uno o más registros de servicio, y recibir mensajes de invocación de servicios web de uno o más solicitantes.
- Un registro de servicios: es responsable de anunciar descripciones de servicios web publicados por los proveedores y permitir a los solicitantes realizar búsquedas en la colección de descripción contenida en el registro. Una vez encontrada la información, el resto de la interacción se realiza directamente entre el solicitante y el proveedor.

1.4 Servicios web

Un servicio web es una aplicación web identificada por un URI[6], cuya interfaz es capaz de ser definida, descrita y descubierta mediante artefactos XML y que soporta la interacción directa con otras aplicaciones software usando mensajes XML y protocolos basados en Internet. Representa una pieza de software que conforma una serie de estándares de intercambio de información. Estos estándares permiten el intercambio de operaciones entre diferentes computadoras, garantizando la independencia del hardware, del software o de los lenguajes de programación que las mismas empleen. Para mantener esta independencia, los servicios web, encapsulan la lógica dentro de un contexto, que puede ser una tarea de negocio, una entidad de negocio o alguna otra agrupación lógica (14).

Son invocados a través de una interfaz y deben cumplir con el principio de reusabilidad. Internamente, su implementación puede contener la ejecución de varios pasos en diferentes aplicaciones que tributen al proceso de negocio y que coordinados den respuesta al servicio en sí. Otra posibilidad es establecer relaciones entre ellos, de forma que su ejecución conjunta cumpla con una funcionalidad en el proceso de negocio. Dicha relación debe ser débilmente acoplada y establecerse a través de mensajes definidos por un estándar. Dado que estos servicios pueden ser consumidos por diferentes sistemas y plataformas, sus características son ideales para implementar una solución con SOA (14).

Las características deseables de un servicio web son (19):

- Debe poder ser accesible a través de la web. Para ello debe utilizar protocolos de transporte estándares y codificar los mensajes en un lenguaje que pueda conocer cualquier cliente que quiera utilizar el servicio.
- Debe contener una descripción de sí mismo, lo cual le permitirá a una aplicación conocer su función y su interfaz, de manera que pueda ser utilizado de forma automática sin la intervención del usuario.

[6] Universal Resources Identifier

- Debe poder ser localizado. Para ello debe contarse con algún mecanismo que permita encontrar un servicio web que realice una determinada función. De esa forma, se posibilitará que una aplicación localice el servicio que necesite de manera automática, sin necesidad de que el usuario lo conozca previamente.

Algunas de las ventajas que ofrecen estos servicios son: (19)

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Al apoyarse en HTTP[7], pueden aprovecharse de los sistemas de seguridad *firewall* sin necesidad de cambiar las reglas de filtrado.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
- Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándares y abiertos.

En el momento de aplicar una SOA uno de los aspectos más preocupantes es la calidad de los servicios. Para lograr y mantener dicha calidad hay que tener en cuenta un grupo de conceptos asociados, cuyo cumplimiento impide el fallo de los servicios. Estos son:

- Política: conjunto de reglas bajo las cuales, un proveedor de servicio hace que el mismo esté disponible para el cliente.
- Administración: conjunto de atributos que pueden aplicarse para manejar los servicios proporcionados o consumidos.
- Transacción: conjunto de atributos que pueden aplicarse a un grupo de servicios para devolver un conjunto de datos consistente.

[7] Hypertext Transfer Protocol

- Seguridad: conjunto de reglas que pueden aplicarse para la identificación, autorización y control de acceso a los consumidores de servicios.

1.5 Estándares para lograr la interoperabilidad

Los estándares son definidos por organizaciones internacionales de estandarización con el objetivo de evitar que intereses privados determinen normas y garantizar la comunicación entre los diferentes dispositivos y/o plataformas con los denominados estándares oficiales. Algunas de estas organizaciones han ofrecido diversas definiciones formales, por ejemplo la ISO[8] lo define *como "acuerdos documentados que contienen especificaciones técnicas u otros criterios, para ser utilizados constantemente como reglas o definiciones de características…"*, por su parte la BSI[9] plantea que los mismos representan *"una especificación publicada que establece un lenguaje común, y contiene una técnica u otro criterio, que está diseñado para ser usado constantemente, como una regla o una definición"* (20).

De manera general, las bibliografías consultadas coinciden en que un estándar es un acuerdo internacional que define reglas o técnicas por las cuales pueden regirse las organizaciones para llevar a cabo sus servicios. Los mismos ofrecen una base de comparación, una medida de calidad y un consenso de opiniones entre individuos, grupos u organizaciones.

1.6 Análisis de soluciones existentes

La era digital ha impulsado a nivel mundial un proceso de informatización que ha conllevado a la inserción de las TIC en casi todas las esferas de la sociedad. Esto ha traído como consecuencia que múltiples empresas hayan comenzado a cambiar la forma en que realizaban muchos de sus procesos internos, adaptándolos a las nuevas tecnologías. Sin embargo, el constante cambio y actualización de esta rama demanda en muchos casos, no el empleo de varios sistemas aislados que trabajen dentro de una misma empresa sin comunicación alguna, sino un único sistema que permita la integración de esos sistemas independientes, de manera que mejore exponencialmente la eficiencia de las

[8] Organización Internacional de Normalización
[9] British Standard Institute

entidades. Son múltiples las propuestas que han surgido para dar solución a este problema. A continuación se explican brevemente algunas de ellas.

Arquitectura de software para la integración de objetos de aprendizaje (OA) basada en servicios web

Se propone una Arquitectura Orientada a Servicios que garantice la interoperabilidad entre los Repositorios de Objetos de Aprendizaje (ROA) y los LMS[10]. La misma cuenta con dos funcionalidades principales: la integración de búsquedas de objetos de aprendizaje (OA) en ROA distribuidos y la construcción de contenidos para plataformas E-Learning, basada en la composición y reutilización de OA. Para el diseño de la misma se empleó el medio de modelado conocido como vista arquitectónica. Así mismo, para su desarrollo se hizo uso del estándar SCORM[11] para la interoperabilidad entre los OA y los ROA, y la implementación de la arquitectura estuvo apoyada en servicios web, debido a que es una tecnología basada en estándares e independiente de la plataforma (21).

S3OiA

S3OiA es una propuesta de arquitectura para la interoperabilidad en el Internet de las Cosas (IdlC), basada en la conocida SOA. La misma permite integrar cualquier tipo de objeto o dispositivo, y utilizar los recursos que ofrecen como base para la generación automática de aplicaciones dinámicas. Estas a su vez, serán capaces de evolucionar y de adaptarse al contexto de ejecución mediante un sistema de gestión de dependencias, basado en eventos, entre dominios remotos. Esta arquitectura basa su comunicación en los servicios web debido a la inexistencia de estándares de interoperabilidad en el Internet del Futuro (IdF). La propuesta se fundamenta en tres niveles de abstracción: un nivel de comunicación físico, un nivel de exposición de servicios localizados en un mismo entorno y una capa virtual de interacción, donde los servicios son agrupados en espacios inteligentes (22).

[10] Learning Management System
[11] Sharable Content Object Reference Model

Modelo Arquitectónico para Interoperabilidad entre Instituciones Prestadoras de Salud (IPS) en Colombia

Se define y valida un modelo arquitectónico de interoperabilidad entre sistemas de información de IPS en Colombia. El modelo sugiere el uso de SOA como referencia para definir la arquitectura de software y el estándar HL7 para el intercambio de mensajes y documentos clínicos. Así mismo, hace uso de otros estándares internacionales para apoyar el intercambio de información asistencial, administrativa y de procesos del negocio, entre los HIS[12] de las diversas IPS en Colombia.

La propuesta de solución alcanza hasta el nivel de interoperabilidad semántica, con el propósito de establecer las bases para alcanzar la interoperabilidad organizacional. Para formalizar el proceso de diseño del modelo arquitectónico, se utilizó como referencia el estándar internacional de la ISO: Reference Model Open Distributed Processing (RMODP), el cual normaliza el desarrollo del modelo de interoperabilidad permitiendo especificar una arquitectura que soporte, principalmente distribución y portabilidad. RM-ODP define cinco vistas para la arquitectura de un sistema: Vista Empresarial, Vista de la Información, Vista Computacional, Vista de la Ingeniería y Vista Tecnológica (23).

Propuesta de arquitectura de interoperabilidad para el intercambio de datos en el Órgano de Justicia-MININT

Propone una arquitectura de interoperabilidad para el intercambio de datos, a partir de la cual se pueda implementar una plataforma informática que contribuya a la eficiencia en el proceso de toma de decisiones, enfocada al ámbito penal. Para lograr el entendimiento entre los sistemas heterogéneos con que cuentan, se siguen las acciones planteadas por la Comisión Económica para América Latina y el Caribe (CEPAL). En el desarrollo de la propuesta de solución se define el uso del estándar XML para el intercambio de datos y el empleo de otros estándares tales como SAML[13], WS-Security, XML Digital Signature y XML Encryption, para lograr mayor robustez en la seguridad, tanto interna como externa (24).

---

[12] Health Information Systems
[13] Security Assertion Markup Language

Conclusiones sobre el análisis de soluciones existentes

Luego del análisis de las soluciones existentes planteadas, puede concluirse que, las mismas están enfocadas hacia la resolución de problemas específicos relativos al ámbito en que se desarrollan. No obstante, las mismas presentan características que pueden ser empleadas como base para el desarrollo de la propuesta de solución. Estas son: el empleo de SOA como referencia para la definición de la arquitectura y el empleo de servicios web, debido a que es una tecnología basada en estándares que garantiza la independencia de la plataforma.

Conclusiones Parciales

El análisis de los diferentes patrones de integración permitió seleccionar los más adecuados para su aplicación en el diseño de la solución propuesta. El estudio del estado del arte permitió identificar a la arquitectura orientada a servicios como la más adecuada para desarrollar la propuesta de solución, así como el empleo de estándares y servicios web como base para llevar a cabo el diseño de la misma.

Capítulo 2: Descripción de la propuesta de solución

En el presente capítulo se presenta la propuesta de solución al problema planteado. Se abordarán los principales aspectos de SOA y se propondrá un ciclo de vida para la arquitectura de interoperabilidad propuesta. Adicionalmente, se realizará una descripción detallada de los componentes arquitectónicos y se profundizará en el estudio de los estándares y protocolos a utilizar.

2.1 Gobernabilidad y ciclo de vida de una SOA

La gobernabilidad no es solo un conjunto de tecnologías que permiten implementar satisfactoriamente una arquitectura SOA, sino también un grupo de herramientas unidas con diversas metodologías, guías y buenas prácticas. Entre estas pueden mencionarse los registros de servicios, las políticas de gestión y de seguridad, los ciclos de vida y los repositorios de información de todo tipo asociada a los servicios, etc. La gobernabilidad en SOA es un tema importante debido principalmente a la interdependencia de los sistemas involucrados, existen múltiples propietarios, múltiples políticas, etc. Ello implica generalmente que cada "incidencia" de reutilización pueda provocar dependencias adicionales.

Según Daniels: *"El ciclo de vida define las fases y las tareas esenciales para el desarrollo de sistemas, sin importar el tipo o la envergadura del sistema que se intenta construir"* (14). Cuando se habla de una SOA, es importante hablar de ciclo de vida, en este sentido, lo más común es identificar 5 fases: *Análisis, Diseño, Desarrollo, Medida (o Monitorización) y Gobierno.*

> ***Análisis:*** comprende normalmente la fase de análisis del proyecto objeto de la arquitectura. Es importante no solo analizar lo existente sino también lo que se va a construir. Es necesario identificar cómo está estructurado el proyecto, las áreas existentes y cómo se encuentra definido, en ese punto debe definirse un Plan de Gobernabilidad correcto y adecuado a las necesidades (que podrá ser modificado si fuese necesario).
>
> ***Diseño:*** una vez identificada la oportunidad para construir la arquitectura SOA deben definirse los distintos niveles que serán necesarios, las personas que van a estar involucradas y en qué medida.
>
> ***Desarrollo:*** se pone en funcionamiento lo diseñado en las fases anteriores. Será necesario aplicar políticas que aseguren la calidad del desarrollo. Se deben registrar los servicios y

hacerse disponibles según el propio ciclo de vida de aceptación para el resto de los usuarios. Además, es en esta fase donde se hace énfasis en el rendimiento de las herramientas tecnológicas que permitirán llevar a cabo una gobernabilidad de la arquitectura de manera correcta.

Monitorización: esta fase permitirá determinar los fallos o deficiencias que tiene la arquitectura, y ofrecer una remodelación de los diseños iniciales. El cumplimiento de los niveles establecidos permitirá controlar los procesos correctos e incorrectos. En este punto se cierra el ciclo de vida y se permite el aseguramiento de la calidad por el esfuerzo de la misma.

Gobierno: para asegurar el éxito de la arquitectura es recomendable crear un centro de excelencia dentro del negocio para implementar las prácticas de gobierno y seguir estándares de control de información y tecnologías relacionadas. Esta fase es crítica para cualquier proyecto SOA.

2.2 Componentes de una SOA

Toda arquitectura está formada por componentes necesarios para el procesamiento de información y para su funcionamiento. A continuación, se describen cada uno de ellos:

Servicio: *e*s la columna vertebral de SOA. Debe aplicar todas las funcionalidades incluidas en los contratos que expone. Debe ser autónomo, lo que significa que el servicio debe ser principalmente autosuficiente. Además, debe proporcionar una función de negocio y ser una parte importante de la lógica.

Contrato: consiste en la colección de todos los mensajes soportados por el servicio. Puede ser unilateral, por tanto ofrece un conjunto cerrado de mensajes que siguen un flujo unidireccional. Alternativamente, un contrato puede ser bilateral, con el servicio de intercambio de mensajes con un grupo predefinido de componentes. El contrato de servicio es análogo a la interfaz de un objeto en el diseño orientado a objetos.

Endpoint: es un identificador universal del recurso URI, expresa la dirección o lugar específico donde se puede encontrar el servicio. Un contrato específico puede ser expuesto en un *endpoint* (punto final) específico.

Mensaje: la unidad de la comunicación en SOA es el mensaje. Los mismos pueden tener diferentes formas, tales como:

- HTTP GET: en la transferencia de estado representacional.
- SOAP: protocolo simple de acceso a objetos.
- SMTP: protocolo para transferencia simple de correo.

La diferencia entre un mensaje y otras formas de comunicación, tales como una llamada a procedimiento remoto[14], no es sutil. Un RPC requiere a menudo que la aplicación que realiza la llamada tenga conocimiento de los detalles de implementación del otro sistema. Con la mensajería, no sucede lo mismo, los mensajes tienen una cabecera y un cuerpo (la carga útil). La cabecera es generalmente genérica y puede ser entendida por los componentes de infraestructura y marco sin conocer los detalles de implementación, esto reduce dependencias y acoplamiento.

Política: un diferenciador importante entre SOA y diseño orientado a objetos es la existencia de políticas. Así como una interfaz o contrato separa especificaciones de las implementaciones, las políticas separan especificaciones dinámicas de las especificaciones estáticas o semánticas. Una política define los términos y condiciones para hacer un servicio disponible para el consumidor. Los aspectos únicos de las políticas son que pueden ser actualizados en tiempo de ejecución y que están externalizados de la lógica de negocio. Una política especifica propiedades dinámicas, tales como la seguridad (encriptación, autenticación, autorización), auditoría, acuerdos de nivel de servicios[15], entre otras.

Consumidor de servicios: un servicio sólo tiene sentido si otro componente de software lo utiliza. Los consumidores de servicios son los componentes de software que interactúan con un servicio a través de mensajería. Los mismos pueden ser aplicaciones cliente u otros servicios; el único requisito es que se adhieran a la SOA en la que intervienen.

[14] RPC por sus siglas en inglés *Remote Procedure Call*
[15] SLA por sus siglas en inglés *Service Level Agreement*

2.3 Estándares utilizados

Algunos ejemplos de estándares abiertos son: XML, XHTML, SGML, SAML, entre otros. Algunos de ellos han logrado el estatuto de norma internacional, sin perder la cualidad de ser una norma abierta. A continuación, se realizará un estudio de los estándares utilizados en la arquitectura propuesta.

SGML

Son las siglas en inglés para *Standard Generalized Markup Language* (lenguaje estándar de marcado generalizado). Según Bastarrica (25), *"sirve para especificar las reglas de etiquetado de documentos y no impone en sí ningún conjunto de etiquetas en especial"*. SGML no es más que un estándar que define los métodos para la representación de información que abarca otros estándares como XML y HTML, entre otros, siendo la raíz de esta gran familia. La ISO[16] normalizó este lenguaje desde 1986, el mismo tiene tres características fundamentales que lo distinguen frente a otros lenguajes de marcación:

- El uso de marcación descriptiva más que procesal. Un sistema descriptivo de marcas usa códigos de marca que simplemente proporcionan nombres para categorizar partes de un documento. Por el contrario, un sistema de marcación procesal define los procedimientos que deben realizarse en los puntos particulares de un documento.
- Su concepto de "tipo de documento". En SGML los documentos se consideran como portadores de tipo, el cual se define formalmente según sus partes constitutivas y su estructura. Si los documentos son tipos conocidos, un programa –analizador SGML- puede usarse para procesar un documento tendente a alcanzar un tipo particular, revisando que todos los elementos requeridos para ese tipo de documento estén realmente presentes y correctamente ordenados.
- Su independencia respecto a cualquier sistema de representación. Los documentos codificados según sus indicaciones pueden ser transportados de una marca de hardware/software a otra sin pérdida de información.

[16] Organización Internacional de Estándares

Capítulo 2: Descripción de la propuesta de solución

XML (*Extensible Markup Language*)

Es un lenguaje abierto que permite describir el sentido o la semántica de los datos, pues separa la presentación del contenido, describiendo este último a través de etiquetas o marcas (25). Por otro lado, Ramón Montero Ayala (26) lo define como *"un lenguaje de marcas basado en SGML, capaz de describir cualquier tipo de información en forma personalizada, aunque también es un metalenguaje de marcado capaz de describir lenguajes de marcas adecuadas para aplicaciones concretas"*, alega además que es "*un conjunto de normas que permiten tratar información muy diversa desde muchos puntos de vista y sistemas diferentes, siendo el propio diseñador el encargado de decidir el proceso más adecuado a cada caso...*"

En resumen, XML es un lenguaje de marcas o de modelado que se encarga más del contenido de los datos que de su presentación. Su estructura se establece a través de etiquetas que marcan su estructura. Además, este estándar puede ser un contenedor de datos y es mucho más flexible que HTML, pues permite definir detalladamente la estructura de un documento sin necesidad de utilizar DTD[17]. Un documento XML tiene varias formas de representación, por tanto, el hecho de que un documento esté en este formato brinda muchas ventajas como portabilidad, depuración, independencia de la plataforma y sobre todo facilidad de edición para posteriores cambios en su estructura. También, XML es toda una familia de tecnologías con distintas especificaciones que cuenta con suficiente soporte y documentación a pesar de ser relativamente nuevo. Una de las razones para optar por XML es que no requiere de licencias, no pertenece a ninguna compañía y es un estándar internacionalmente reconocido (27).

XML Schema

Es un documento de definición estructural al estilo de los DTD, que además cumple con el estándar XML y permite expresar mayor diversidad de documentos (25). Los documentos XML-Schema (generalmente con extensión XSD) se concibieron como un sustituto de los DTD, teniendo en cuenta los puntos débiles de estos y buscando mejores capacidades a la hora de definir estructuras para los documentos XML, como la declaración de los tipos de datos. En la Universidad de las Ciencias

[17] Document Type Definition

Informáticas, se definen los documentos XSD como *"la descripción de la estructura de la información contenida en un archivo XML y de sus reglas. Se requiere para que los sistemas que transmiten o reciben archivos XML puedan validar la conformación de estos archivos o reglas definidas por los autores"* (28).

WSDL (*Web Service Description Language*)

El lenguaje de descripción de servicios web es un dialecto basado en XML sobre el esquema que describe un servicio web. WSDL es extensible y se puede utilizar para describir, prácticamente cualquier servicio de red. Según Tortosa (20), *"es un formato XML que describe los servicios de red como un conjunto de endpoints que procesan mensajes contenedores de información orientada tanto a documentos como a procedimientos"*. Otros autores como Francisco Domínguez Mateos (28), plantean que es *"básicamente información XML en la que se aporta una descripción de las interfaces que intervienen en determinados servicios junto con los protocolos de invocación que soporta"*. Un documento de este tipo proporciona la información necesaria al cliente para interactuar con el servicio web.

Para la definición de un servicio se utilizan los siguientes elementos:

- *Types:* contenedor de definiciones del tipo de datos que utiliza según el sistema de tipos.
- *Message:* definición abstracta y escrita de los datos que se están comunicando.
- *Operation:* descripción abstracta de una acción admitida por el servicio.
- *PortType:* conjunto abstracto de operaciones admitidas por uno o más puntos finales.
- *Binding:* especificación del protocolo y del formato de datos para un tipo de puerto determinado.
- *Port:* punto final único que se define como la combinación de un enlace y una dirección de red.
- *Service:* colección de puntos finales relacionados.

WSDL distingue claramente los mensajes de los puertos: los mensajes (la sintaxis y semántica que necesita un servicio web) son siempre abstractos, mientras que los puertos (las direcciones de red en las que se invoca el servicio web) son siempre concretos. No es necesario que un archivo WSDL incluya información sobre el puerto, puede contener simplemente información abstracta de interfaz, sin facilitar datos de implementación concretos y ser válido. De este modo, los archivos WSDL se separan de las implementaciones.

SOAP (*Simple Object Access Protocol*)

Es un protocolo ligero de mensajes XML que se usa para recoger información de los mensajes de petición y respuesta de los servicios web que se envían a través de la red. Según Tortosa (20), *"es un estándar que define un protocolo que da soporte a la interacción (datos + funcionalidad) entre aplicaciones en entornos distribuidos y heterogéneos, es interoperable (neutral a la plataforma, lenguajes de programación, independiente del hardware y protocolos). Define como organizar la información de una manera estructurada para intercambiarla entre los distintos sistemas"*. Por otra parte Naranjo (29) lo define como *"una arquitectura conceptual que organiza funciones de negocio como servicios interoperables permitiendo reutilización de servicios para satisfacer necesidades de .negocio."*

SOAP está compuesto de tres partes:

- Un envoltorio que define un marco de trabajo que describe qué existe en el mensaje y cómo procesarlo.
- Un set de reglas de codificación para expresar instancias de tipos definidos en una aplicación.
- Una convención para representar llamadas y respuestas a procedimientos remotos.

Los mensajes SOAP son independientes de los sistemas operativos, pueden ser transportados usando una variedad de protocolos, por ejemplo, SMTP y HTTP. Los mismos definen cómo realizar la codificación de las llamadas a los métodos de un servicio web, y cómo debe el servicio web codificar el resultado para que pueda ser interpretado. El mensaje está contenido en un sobre (*envelope*) que debe ser siempre la primera sección del mensaje, el cual se compone de una cabecera (*header*) que es opcional, y de un cuerpo (*body*) que es obligatorio. La cabecera se utiliza para enviar información sobre identificadores de transacciones, certificados de seguridad, información sobre coordinación, etc. En caso de estar presente debe ser el primer hijo de la construcción *envelope* (19).

El cuerpo contiene las llamadas a los procedimientos remotos junto con sus parámetros, o bien las respuestas de dichos procedimientos remotos, o información sobre el error que se haya producido. Los usos típicos de esta construcción son proveer un mecanismo simple para intercambiar información con el receptor del mensaje SOAP.

Capítulo 2: Descripción de la propuesta de solución

REST

Representational State Transfer, que traducido al español significa transferencia de representación de estados. Es un estilo de arquitectura de software para construir aplicaciones distribuidas basadas en los principios que hicieron exitosa la web, modelando un servicio como la aplicación de una serie de operaciones fijas sobre un conjunto de recursos posibles de referenciar universalmente. Está fuertemente basado en HTTP, porque aunque es independiente de este protocolo, es el único utilizado masivamente para soportar los principios REST (30).

En el documento que contiene los lineamientos de arquitectura para los sistemas informáticos que pertenecen a la intranet de las Universidad de las Ciencias Informáticas, se plantea que REST *"es un estilo arquitectónico y no un estándar definido. Se basa en las URI para identificar recursos, los cuales se retornan en XML puro, es también una manera simplificada de ejecutar métodos remotos"* (28).

Un servicio web REST tiene las siguientes características:

- Las interfaces deben construirse sobre HTTP. Definiendo las siguientes funciones:
  - HTTP GET: usado para obtener una representación de un recurso. Un consumidor lo utiliza para obtener una representación desde una URI. Los servicios ofrecidos a través de esa interfaz no deben contraer ninguna obligación respecto a los consumidores.
  - HTTP DELETE: se usa para eliminar representaciones de un recurso.
  - HTTP POST: se usa para actualizar o crear las representaciones de un recurso.
  - HTTP PUT: se usa para crear representaciones de un recurso.
- La mayoría de los mensajes son XML.
- Mensajes simples que se pueden codificar en las URL´s.
- Los servicios y los proveedores de servicios deben ser recursos, mientras que los consumidores pueden ser un recurso.

De manera general los servicios web REST necesitan poca infraestructura, aparte de las tecnologías HTTP estándar y XML, que actualmente son soportadas por la mayoría de los lenguajes de programación y plataformas. Son simples y efectivos, ya que HTTP es el interfaz más extendido y es soportado por la mayoría de las aplicaciones.

Capítulo 2: Descripción de la propuesta de solución

Al realizar un análisis de SOAP y REST, se puede apreciar que es más factible optar por SOAP, pues REST está fuertemente arraigado al protocolo HTTP, imposibilitando la utilización de protocolos como FTP[18] y SMTP. La seguridad depende del protocolo HTTPS[19], por lo que si se evita la misma, se compromete el sistema. En SOAP los mensajes se representan en estructuras XML bien definidas, mientras que en REST se encuentran en XML planos. Otro factor importante es que en SOAP los contratos o interfaces son definidos a través de WSDL utilizando esquemas XML, mientras que REST no posee un método formal para expresar dichas interfaces (30).

Sin embargo, en la presente investigación se define la utilización de ambos indistintamente, permitiendo una mayor interoperabilidad e integración entre los componentes, debido a que la arquitectura contará con ambos para el intercambio de mensajes.

UDDI (*Universal Description, Discovery and Integration*)

Varios autores hacen referencia a este estándar definiéndolo como *"un directorio que contiene un registro o repositorio de descripciones de servicios web. Tiene dos objetivos fundamentales, servir de soporte a los desarrolladores para encontrar información sobre servicios web y poder construir un cliente y facilitar el enlace dinámico de servicios web permitiendo consultar referencias y acceder a servicios de interés"* (20). Por otro lado, Pedro Espina Martínez (31), plantea que UDDI está comúnmente considerado en la actualidad como la piedra angular de las arquitecturas SOA definiendo un método estándar de publicación y descubrimientos web.

De manera general, se puede afirmar que UDDI es un estándar que permite gestionar un registro o repositorio de servicios web, en cuyo contenido se encuentran las inscripciones, interfaces o contratos de dichos servicios para su localización y acceso automático. El mismo se apoya en algunos estándares que han sido muy aceptados, tales como XML y SOAP entre otros. Consiste en ver qué datos se almacenan y cómo se estructuran.

La información en UDDI se almacena en nodos, que pueden ser privados o públicos. En la actualidad existen dos nodos públicos muy utilizados mundialmente que son los de Microsoft e IBM. Es

[18] File Transfer Protocol
[19] Hypertext Transfer Protocol Secure

importante destacar que no existen requisitos de implementación para estos nodos, ya que se puede usar cualquier lenguaje o herramienta. Es relativamente ligero, se ha diseñado como registro, no cómo depósito; la diferencia, aunque es sutil, resulta esencial. Un registro redirige al usuario a recursos, mientras que un depósito solo almacena información (32).

Todos los datos en el repositorio deben pertenecer a uno de los cuatro tipos de datos; los cuales se definen en una estructura de datos basada en XML y cada una contiene campos obligatorios y opcionales. Los cuatro tipos de datos base son (32):

- *businessEntity:* captura la información sobre un negocio o entidad, la cual es utilizada por el negocio para publicar información descriptiva sobre sí mismo y los servicios que ofrece.
- *businessService:* representa los servicios o procesos de negocios que provee la estructura *businessEntity.*
- *bindingTemplate:* representa los datos importantes que describen las características técnicas de la implementación del servicio ofrecido.
- *tModel:* representa una especificación técnica.

Lo anterior puede clasificarse en páginas amarillas, blancas y verdes como se muestra en el Anexo 1 de esta investigación.

El operador del repositorio UDDI genera un identificador único para cada tipo de datos cuando la información es publicada por primera vez en el repositorio. El Anexo 2 muestra la interrelación entre los cuatro tipos de datos y su clasificación.

Para resumir lo descrito anteriormente, el proceso técnico para descubrir, encontrar e invocar un servicio web es:

- Usar el registro UDDI para localizar el elemento *businessEntity* del negocio apropiado.
- Localizar el elemento *businessService* para identificar los servicios web ofrecidos por la empresa.
- Seleccionar el *bindingTemplate* para recuperar la dirección del servicio web y el elemento *tModel* para asegurar la compatibilidad técnica entre los sistemas.

Existen diversos protocolos que permiten gestionar la seguridad a distintos niveles tales como el cifrado de datos y documentos, el aseguramiento de las transacciones entre los servicios web, entre

otros. A continuación, se mencionan algunos de los que se propone incorporar a la arquitectura propuesta.

XML-Encryption

Es un lenguaje cuya función principal es asegurar la confidencialidad de partes de documentos XML a través de la encriptación parcial o total de un documento transportado. Como ventaja tiene que se puede aplicar a cualquier servicio web. También permite granular el cifrado de documentos XML haciendo posible el cifrado a varios niveles que van desde elementos básicos hasta documentos completos (33). Algunos de estos niveles son:

- Cifrado de un elemento.
- Cifrado de elementos hijos: solo se cifran datos específicos de un elemento.
- Cifrado de contenido textual: se puede cifrar el contenido textual de un elemento.
- Cifrado de documentos completos.
- Súper cifrado: se cifran datos que ya hayan sido cifrados, es decir, es un doble cifrado.

XML Digital Signature

Es un estándar que provee mecanismos para la creación de firmas digitales a partir de XML. Asegura la integridad de partes de un documento y define un esquema para capturar el resultado de la operación de firmas digitales y aplicarlas a los metadatos. Proporciona además la autenticación de mensajes y/o servicios de autenticación de firma para datos de cualquier tipo. En resumen, asocia claves con los datos de consulta y representa un sistema que a través de una firma digital ofrece la autenticidad de los datos y el mensaje, así como su integridad (34) (35).

Web Service Security

Es una especificación encaminada a brindar seguridad para SOAP, proveyendo seguridad para los mensajes involucrados en transacciones de servicios web. El mismo no define prácticamente ninguna nueva tecnología de seguridad, sino que se enfoca en la correcta y efectiva aplicación de las tecnologías existentes. Aprovecha la flexibilidad de XML Digital Signature y XML-Encryption permitiendo asegurar selectivamente partes de los mensajes. Con servicios web como un *firewall*, puede leer las partes del mensaje que necesita; esto ha resultado imposible con tecnologías anteriores que solamente entregan un canal de comunicación seguro.

De manera general, es un protocolo de comunicación que provee los medios necesarios para aplicar seguridad a los servicios web, contiene especificaciones que de cierta forma garantizan la integridad y seguridad en mensajerías entre servicios web. Además, permite el uso de tecnologías de encriptación como las PKI[20] y el protocolo SSL[21]/TLS[22], que son elementos claves a tener en cuenta siempre que se quiera garantizar seguridad en cualquier sistema informático (36).

SAML (*Security Assertion Markup Language*)

Proporciona un idioma común para el despliegue de toda la infraestructura de seguridad. Permite intercambiar información de identificación y está basado en XML. Proporciona un protocolo de mensaje tipo pregunta-respuesta para que los dominios implicados puedan gestionar las decisiones necesarias de autenticación y autorización, esto puede ser aplicado a los servicios web para proveer seguridad a las transacciones de los mismos. (37) (38).

2.4 ESB

Una de las partes más importantes de SOA es el ESB[23] ya que es el mediador multiprotocolo y multipropósito para dicha arquitectura. Es una infraestructura compuesta por diversos servicios dentro de los que se encuentra el servicio de mensajería, enrutamiento inteligente, auditorías, autorización, transporte de datos y conversión de mensajes (36).

Las aplicaciones y características más comunes del ESB son las siguientes (39):

- *Enrutamiento basado en contenidos:* generalmente, el ESB cumple un cometido de enrutación entre sistemas o aplicaciones. Para ello, el mensaje debe llevar la información asociada que permita determinar el punto final (*endpoint)* en cuestión (es lo que suele llamarse metadatos o metainformación del mensaje).

[20] Public Key Infraestructure
[21] Secure Sockets Layer
[22] Transport Layer Security
[23] Enterprise Service Bus

- *Transformación de mensajes:* la mayoría de los ESB suelen incorporar motores de transformación y parseadores de mensajes que permiten y facilitan la transformación de los mismos.
- *Configuración y no codificación:* una de las características más ventajosas de un ESB es que suele realizar sus funciones por medio de configuración y no de codificación. Actualmente la mayoría incluye un Entorno de Desarrollo Integrado[24] gráfico que facilita la configuración de los elementos del ESB.
- *Proxy de servicios:* una de las mayores ventajas del ESB es la de crear una capa de proxy por encima del servicio. Ello permite abstraer su implementación (que puede ser como una caja negra, de la que solo se dispone de la interfaz, normalmente su WSDL) y añadirle información o requisitos para su ejecución.
- *Conversión de protocolos:* una de las características del ESB es su funcionalidad multiprotocolo. Esto permite la conexión a distintas aplicaciones o sistemas por medio de numerosos protocolos.
- *Auditorías y logs de mensajes:* la comunicación de información a través del ESB permite centralizar sistemas de *log (registro)* y persistencia de información susceptible de ser auditada. Además, con el uso de diferentes herramientas pueden explotarse dichos datos y realizar monitorizaciones exhaustivas que permitan la optimización del ESB, la arquitectura y de manera general del negocio de la empresa.
- *Manejo de excepciones:* generalmente los servicios web tienen un comportamiento de respuesta correcta y en algunos casos pueden devolver excepciones de manera controlada o no.
- *Acceso a repositorio de servicios:* un repositorio de servicios permite acceder a metainformación de los servicios integrados, lo que facilita el control de los mismos y el poder de aplicar políticas y condiciones de una manera centralizada y jerárquica.
- *Validación, Enriquecimiento, Transformación y Operación de Mensajes:* se puede decir, que estas son las cuatro características esenciales de un ESB:

[24] IDE por sus siglas en inglés Integrated Development Environment

- ✓ Validación de la información: validación de los mensajes entrantes, sobre los que se pueden aplicar distintas condiciones y verificaciones que hagan cumplir los requisitos establecidos.
- ✓ Enriquecimiento de los mensajes: posibilidad de añadir información o metainformación adicional para cumplir las necesidades de integración en cada momento.
- ✓ Transformación de los mensajes: normalmente entre los diversos modelos de datos del conjunto de servicios y aplicaciones a integrar.
- ✓ Operación de los mensajes o enrutación en función de directivas preestablecidas o a partir de metainformación incluso en la propia comunicación.

2.5 BPM

Se propone también como componente de la arquitectura un BPM[25]. El mismo representa el conjunto de servicios y herramientas que facilitan la administración de procesos de negocio. Por administración de procesos se entiende: análisis, definición, ejecución, monitoreo y control de los procesos. El mismo contempla un soporte para interacción humana, e integración de aplicaciones, siendo esta, su diferencia fundamental con la tecnología de *WorkFlow* (Flujo de trabajo), ya que BPM integra en los flujos a los sistemas. (40).

BPM también es vista como una disciplina de administración, que requiere que las organizaciones se cambien a un pensamiento centrado en los procesos y que reduzcan su dependencia de estructuras tradicionales de territorio y funcionalidad.

¿Existen riesgos al utilizar un ESB fuera del componente BPM? Generalmente se tienden a confundir las capas de *middleware* con las reglas de negocio, algo similar ocurre cuando los servicios web compuestos se vuelven muy pesados y terminan siendo más que compuestos. Esto significa que comienzan a manejar parte de la capa lógica de negocio. BPM será el componente que deberá concentrar los procesos, las reglas y la lógica del negocio. El rol de un ESB no trata con lógica de

[25] Business Process Management

negocio, trata con lógica y monitorización de servicios, acceso a servicios, homogeneización de los mismos, administración de adaptadores, etc.

2.6 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que un sistema debe cumplir. Estos requerimientos dependen del tipo de software que se desarrolle, de los posibles usuarios del mismo y del enfoque general tomado por la organización al redactar los requerimientos. A continuación, se definen los requisitos funcionales que facilitan la arquitectura propuesta:

RF1- Crear plan de actividades a nivel provincial.

RF2- Mostrar plan de actividades mensual.

RF3- Mostrar las actividades de un usuario determinado.

RF4- Consultar miembro.

RF5- Modificar y eliminar miembro.

RF6- Enviar al eXcriba los datos de la solicitud de inscripción.

RF7- Mostrar el estado del presupuesto.

RF8- Mostrar estado de la cotización por persona.

RF9- Mostrar el estado de la cotización por municipio.

RF10- Mostrar el estado de la cotización por provincia.

2.7 Requisitos no funcionales

Los requisitos no funcionales son las cualidades o propiedades que el producto debe tener y que imponen restricciones en el diseño. Son fundamentales en el éxito del producto y tienen una estrecha vinculación con los requisitos funcionales. Son la base de la arquitectura de software, ya que una vez que se conozca lo que el sistema debe hacer, es posible determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. A continuación se describen los requisitos que el sistema debe cumplir:

Seguridad

- El acceso a cualquier manipulación del sistema debe estar sometido a un proceso de autenticación del usuario donde será especificado el usuario y la contraseña.
- Las contraseñas deberán tener más de 7 caracteres de longitud y una fortaleza media.
- Los usuarios estarán obligados a cambiar la contraseña cada 60 días como máximo, pueden cambiar todos los días si lo desean.
- Paralelo a la base de datos primaria se debe mantener una base de datos que registre todas las modificaciones hechas a la original, ordenadas cronológicamente y con la especificación del usuario responsable de dicha modificación, de manera que siempre se realicen trazas a la información manejada.
- Se debe garantizar comunicaciones seguras entre los clientes y el servidor, encriptando todo el tráfico de información con la utilización de llaves negociadas, algoritmos y protocolos.

Extensibilidad

- Se debe lograr un diseño adaptable, con la capacidad de soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

Disponibilidad

- Se debe garantizar el funcionamiento de la aplicación durante las 24 horas del día y los 7 días de la semana con el menor tiempo posible de recuperación de fallos.

Software

- Sistema Operativo: Linux, Solaris o Windows.
- Navegadores: Mozilla Firefox 3.5, Google Chrome 8.0, Internet Explorer 6 o superior.
- Oracle Java SE Development Kit (JDK) 1.6.*/1.7.* o superior.
- Apache ActiveMQ JMS Provider.
- Apache Ant.
- Apache Maven.

Hardware

- RAM: Mínimo 2GB.
- Tamaño de la pila: 512MB.

- Disco: 180MB excluyendo el espacio reservado para ficheros de *logs* y bases de datos.

Restricciones en el diseño y la implementación

- El lenguaje de programación que se debe utilizar es Java.
- El gestor de base de datos es PostgreSQL y/o Microsoft SQL Server.

2.8 Arquitectura de interoperabilidad propuesta

Desde el punto de vista tecnológico, SOA propone varias capas de servicios que exponen funcionalidad, fundamentalmente de negocio, que permiten la composición de aplicaciones a partir de los mismos. La SOA estará constituida por varias capas:

- Capa de Presentación: será la encargada de interactuar con los usuarios, expondrá los datos necesarios y se encargará de la manipulación de entrada y recepción de datos.
- Capa de Servicios: Esta capa puede ser subsidiada o subdividida en varias capas, alberga lógica de negocio, seguridad, metadatos, etc.
- Capa de Acceso a Datos: se encargará de todo el procesamiento de datos que necesiten las capas superiores.

Como toda arquitectura que siga el patrón de capas, cada una tendrá su nivel de responsabilidades. Las capas superiores solicitarán servicios a las capas inmediatamente inferiores. Este paradigma de estructura por capas proporciona las ventajas de reutilización, flexibilidad, mantenimiento y escalabilidad. Permite además dividir proyectos complejos y largos en varios más pequeños y a su vez más fáciles y seguros de elaborar.

Capa de Presentación

Es la capa con la que interactúa el usuario, permitiendo comunicar y capturar la información del mismo en un mínimo de proceso. Esta capa se comunica únicamente con el ESB, el eXcriba, o con el BPM Bonita. También es conocida como interfaz gráfica y debe tener la característica de ser

entendible y fácil de usar para el usuario. Para esta capa se propone el portal de la UIC que está desarrollado en el CMS[26] LifeRay.

Liferay o formalmente Liferay Portal es una aplicación *software* que permite la creación de portales empresariales incluyendo funcionalidades de gestión de contenido, mensajería instantánea, correo electrónico, encuestas y calendarios compartidos, entre otras. Ofrece una intuitiva interfaz web que permite la configuración y personalización de distintos apartados para la creación de un completo portal empresarial, permitiendo que éste pueda ser creado prácticamente sin tener que escribir código (41).

Los *portlets* son pequeñas aplicaciones web escritas en Java que se ejecutan en una parte de una página web, formando el corazón del portal, ya que son los que contienen todas sus funcionalidades. Estas mini-aplicaciones pueden ser locales o remotas al portal, una de sus principales características es que son totalmente autónomos. Estos pueden ser desarrollados utilizando cualquier *framework* de java que soporte el desarrollo de *portlets* (41).

Capa de Servicios

La capa de servicios de una arquitectura SOA tiene la responsabilidad de modelar la lógica de negocios, ofrecer funcionalidad a la aplicación, minimizar dependencias entre el negocio y la aplicación a través de los servicios y reutilizar estos servicios de negocio en un ESB. Esta capa además proporciona servicios a la Capa de Presentación en dependencia de las solicitudes del usuario. En la misma se propone la utilización del BPM Bonita, el WSO2 ESB, el WSO2 AS y el WSO2 GREG. A continuación, se realiza un estudio de las principales características de dichas herramientas:

BPM Bonita 7.0.3

Bonita *Open Solution* (Solución Abierta Bonita) es una *suite* para la gestión de procesos de negocio y realización de *workflows*, creada en 2001. Es código abierto y puede ser descargado bajo GPL[27] v2.

[26] Sistema de Gestión de Contenidos
[27] General Public License

Se utiliza para modelar gráficamente un proceso de negocio con la notación BPMN[28] y generar procesos que permitan automatizar los procesos de la organización. Se puede ejecutar bajo plataforma Linux o Windows. Cuenta con un diseñador de procesos intuitivo y gráfico, el cual permite diseñar formularios, de forma sencilla.

WSO2 ESB

WSO2[29] es una compañía que ha desarrollado un conjunto de herramientas de código abierto basado en estándares que permite la adopción tecnológica de SOA. Su plataforma está basada en componentes usando la tecnología OSGI[30] y ha sido ubicado como visionario en las suites de SOA. Entre las razones de mayor peso para su elección se encuentra su nivel de cumplimiento con los estándares analizados para el desarrollo de servicios web, su tipo de licencia (Apache v2), la calidad de su documentación, su característica de multiplataforma, sus facilidades de trabajo en entornos que necesitan alta disponibilidad y su buen rendimiento (42).

El Bus de Servicios Empresariales de WSO2 es un producto ligero, de alto rendimiento y prácticamente sin latencia. Provee soporte para tecnologías SOAP, servicios web y servicios REST; también es compatible con todos los patrones de integración empresarial. Es completamente configurable desde su interfaz de usuario y sus capacidades son extensibles debido a la gran cantidad de conectores que incorpora (43).

El ESB de WSO2 está basado en Apache Synapse. Soporta los principales estándares para el transporte y entrega: HTTP, HTTPS, POP[31], IMAP[32], etc. Brinda soporte a los principales formatos de mensaje como JSON[33], XML, SOAP, WS-*, REST, entre otros. Realiza el enrutado de mensajes sobre cabeceras, contenidos de mensajes y basado en reglas.

WSO2 AS

---

[28] Business Process Model and Notation
[29] Web Service Oxygen
[30] Open Source Gateway Initiative
[31] Post Office Protocol
[32] Internet Message Access Protocol
[33] JavaScript Object Notation

Capítulo 2: Descripción de la propuesta de solución

El Servidor de Aplicaciones de WSO2, posee una interfaz gráfica que permite la integración con un IDE. Es una herramienta para el almacenamiento de servicios desarrollados en diferentes *frameworks* de desarrollo como Axis, Axis2, JAX-WS, Servicios Spring y JAR; también sirve para desplegar aplicaciones web tradicionales y servicios *RESTful*. Las aplicaciones web desplegadas en esta plataforma pueden compartir la lógica de negocio, datos y procesos a través de todo el ecosistema de herramientas. Permite la realización de pruebas funcionales, tiene amplias funcionalidades de configuración de servicios, pues ella misma genera los WSDL de los servicios web. Permite adoptar mecanismos de restricción de acceso a servicios a través de filtros por direcciones IP o por subredes y dominios. Se integra con WSO2 Governance Registry para la gestión de ciclos de vida de los componentes desplegados. Agrupa las mejores tecnologías de código abierto para aplicaciones web, habilitando extensiones de monitoreo, clusterización y otras prestaciones (44).

WSO2 GREG

El Registro de Gobierno WSO2 proporciona el nivel adecuado de estructura para apoyar la gobernabilidad de SOA, la configuración de gobierno, el gobierno del proceso de desarrollo, el gobierno en diseño y en tiempo de ejecución, la gestión del ciclo de vida y la colaboración en equipo. También ofrece ahorro de tiempo significativo y adquisición asequible. Especialmente diseñado para la rápida configuración y extensión eficiente.

El GREG permite gestionar los metadatos de los servicios desde su identificación, crear múltiples ciclos de vida a través de un XML, gestionar los diferentes estados de los servicios definiendo los requerimientos a cumplir para ir de un estado a otro. Notifica de manera instantánea sobre cambios en los estados de los servicios, expone los servicios web vía UDDI o usando WS-Discovery y permite de conjunto con el resto de las herramientas, la publicación de servicios web implementados o desplegados en el AS y el ESB, los cuales se hacen visibles en el GREG y se pueden gestionar desde ahí. (45).

Gestor de Documentos Administrativos eXcriba

El Gestor de Documentos Administrativos (GDA) eXcriba está orientado a cualquier entidad que genera documentos como prueba y testimonio de sus actos administrativos, permitiendo facilitar la gestión eficiente de sus documentos. Es una solución de software desarrollada en la UCI que apoya el control y la organización de la información que se genera o recibe.

Capítulo 2: Descripción de la propuesta de solución

El eXcriba tiene como núcleo al ECM[34] Alfresco y ofrece una interfaz para poder llevar a los clientes las bondades que este ECM provee como repositorio documental. Alfresco es compatible con sistemas operativos tales como: Microsoft Windows, Linux, Unix y está desarrollado en Java. (46).

Alfresco en su arquitectura, ofrece una capa dedicada a brindar los servicios web que permiten obtener el contenido que se almacena en el repositorio y facilitan la integración con otras aplicaciones externas. Dicha capa se basa en una API de servicios web disponible para el acceso al repositorio, diseñada para facilitar su comprensión y utilización, es accesible a tantos lenguajes como sea posible y proporciona acceso remoto al repositorio.

A continuación, se mencionan los distintos conjuntos de servicios web que implementa Alfresco para dar acceso a las distintas funcionalidades de su repositorio: (47)

- Autenticación (*Authentication*): conexión y desconexión.
- Repositorio (*Repository*): consulta y manipulación del modelo.
- Contenido (*Content*): manipulación del contenido.
- Autoría (*Authoring*): creación colaborativa de contenido.
- Clasificación (*Classification*): aplicación de clasificaciones y categorías.
- Control de acceso (*Access Control*): roles, permisos y propiedades.
- Acciones (*Action*): gestión de acciones y reglas.
- Administración (*Administration*): gestión de usuarios, exportación e importación.
- Diccionario (*Dictionary*): descripción de modelos.

Cada uno de los servicios web se basa en los siguientes tipos de datos para la entrada y salida de mensajes: (47)

- Identificadores (*Identifiers*): medios para la identificación y localización de contenido.
- Contenido (*Content*): datos de los contenidos.
- Consulta (*Query*): consultas y conjuntos de resultados.
- Metadatos (*Meta Data*): metadatos de los contenidos (diccionario de datos).

[34] Enterprise Content Management

- Versionado (*Versioning*): histórico y gráficos de versiones.
- Clasificación (*Classification*): categorización de contenidos.

Para darle solución a los requisitos funcionales relacionados con el GDA eXcriba, RF4, RF5, RF6 se propone el servicio web para la manipulación de contenidos. Dicho servicio implementa los siguientes métodos:

***read*:** lee el contenido de un tipo de dato incluido en un nodo. Recibe como parámetros los nodos que se deben leer y el nombre del tipo de dato contenido. Devuelve los datos relativos a los nodos de tipo contenido.

***write*:** escribe el contenido en el repositorio en un nodo de tipo contenido. Recibe como parámetros la referencia al nodo, el nombre del tipo de dato, el contenido y el formato del contenido. Devuelve los datos relacionados con el nodo de tipo contenido escrito.

clear: elimina el contenido de un nodo de tipo contenido. Recibe como parámetros un nodo de tipo contenido y el nombre del tipo de dato. Devuelve los datos relacionados con el nodo de tipo contenido que se elimina.

Sistema de Planificación de Actividades

El Sistema de Planificación de Actividades (SIPAC) desarrollado en la UCI, está enfocado a facilitar la gestión de las actividades a todos los niveles organizacionales. Permite interrelacionar objetivos de trabajo y actividades en tiempo real, garantizando el seguimiento del desarrollo y cumplimiento de los objetivos y tareas principales de las entidades. Así mismo, permite informatizar los procesos de elaboración, ejecución y control del plan, evaluación de los objetivos y puntualización de las actividades (48).

El SIPAC está compuesto por los módulos: Configuración, Planificación, Recuperaciones y Notificaciones. Estos módulos son los encargados de la ejecución del proceso de planeación estratégica y operativa en el sistema de planificación (48).

Módulo Configuración

Contiene las funcionalidades para el trabajo con las configuraciones de grupos de usuarios por rol y permisos y los diferentes nomencladores.

Módulo Planificación

Contiene las operaciones a realizar con los documentos de la planificación.

Módulo Recuperaciones

Abarca una serie de reportes concebidos para efectuar el análisis de la planificación en un período o en un rango de fecha determinado.

Módulo Notificaciones

Permite generar además de las notificaciones, los reportes de acciones realizadas sobre elementos a los cuales el usuario tiene acceso, a partir de determinadas acciones que se realizan sobre los diferentes elementos.

Actualmente el Sistema de Planificación de Actividades no cuenta con servicios web definidos que permitan ser consumidos desde otros sistemas, es decir, no cuenta con un repositorio de servicios web ni con interfaces o contratos descritos. Sin embargo, se identificó una manera de solucionar el RF3- Mostrar las actividades de un usuario determinado, utilizando el estilo de arquitectura REST y como protocolo de transporte HTTP. La solución consiste en enviar una petición a la URL 'http://sipac.uci.cu/planificacion/pdo/plan/index.php/cmpcalendario/cargarelementoscalendario'.

El SIPAC debe recibir como parámetros: el identificador del calendario y el del usuario, rango de fecha y un arreglo vacío para cada uno de los siguientes atributos: categoría de las actividades, tipos de actividades y elementos que tributan a la actividad. Finalmente, el sistema retorna por HTTP un archivo JSON con el asunto de la actividad, la descripción, el lugar de la actividad, la fecha de inicio y de cierre. Este requisito funcional se implementa directamente en un *portlet*. Como se explica anteriormente, los mismos son desarrollados en Java, y dicho lenguaje brinda soporte para el tratamiento de datos en formato JSON y el estilo REST.

Sistema de Planificación de Recursos Empresariales Odoo

Odoo en su versión 8.0 es el marco de desarrollo utilizado para construir sistemas de gestión empresarial (ERP), de código abierto y sin coste de licencias que cubre las necesidades de las áreas de: Contabilidad y Finanzas, Ventas, Recursos Humanos, Compras, Proyectos, Almacenes, CRM[35] y Fabricación (49). En la UCI actualmente se trabaja en la migración hacia el ERP Odoo por las

[35] Customer Relationship Management

funcionalidades que brinda y el nivel de aceptación que tiene a nivel mundial en el campo empresarial. Sus principales características son: (49)

- Es un sistema de código abierto.
- Es multiplataforma.
- Fácil manejo.
- Posee una importante comunidad de desarrolladores.
- Integración con otras aplicaciones.

El servidor Odoo proporciona una API *Web Service* que se utiliza por su cliente web y también está disponible para otras aplicaciones, permitiendo acceder a muchas de sus características. Todos sus datos están disponibles para la integración o análisis con diversas herramientas. La API *Web Service* Odoo se puede acceder externamente utilizando dos protocolos diferentes XML-RPC y JSON-RPC. Cualquier programa externo capaz de implementar un cliente para uno de estos protocolos será capaz de interactuar con un servidor de Odoo. En base a los requisitos no funcionales mencionados anteriormente se propone utilizar el lenguaje de programación Java.

Para darle solución a los requisitos funcionales relacionados con el ERP Odoo, RF7, RF8, RF9 y RF10, se proponen utilizar las siguientes funcionalidades que brinda la API *Web Service* de Odoo:

Conexión (*Connection*): permite la conexión directa a un servidor Odoo instalado y configurado previamente. Recibe como parámetros la URL del servidor Odoo, el nombre de la base de datos, el usuario y la contraseña con rol de administrador en el servidor.

Leer registros (*Read records*): permite leer datos del registro mediante el método *read()*. Por defecto devolverá todos los campos que el usuario actual puede leer.

Listado de campos de registro (*Listing record fields*): el método *fields_get()* puede ser utilizado para obtener todos los campos de un modelo y comprobar los que sean de interés para el usuario.

Es importante señalar, que la utilización de manera aislada, de los servicios que exponen los sistemas anteriores no ofrece solución a los requisitos funcionales declarados anteriormente. Sin embargo, la arquitectura brinda la posibilidad de desarrollar servicios web que puedan ser declarados de acuerdo a los estándares y protocolos que se proponen haciendo uso de los componentes WSO2

AS y WSO2 ESB, y apoyándose en los servicios web de los sistemas estudiados que se desean interoperar. Permitiendo además que los servicios implementados puedan ser utilizados por sistemas externos.

Capa de Acceso a Datos

Esta capa tiene la función de realizar los accesos hacia la base de datos para la persistencia de la información, así como también para las consultas de la misma. Generalmente los datos se guardan en un sistema gestor de bases de datos relacional. Para la persistencia se usan componentes de acceso a datos.

Para la propuesta de arquitectura se plantea utilizar la infraestructura desplegada actualmente en la UIC. La cuál está conformada por dos clústeres de base de datos. Un clúster es un conjunto de computadoras construidas mediante la utilización de hardware común y que se comportan como si fuese una sola computadora; convirtiéndolo en un ordenador más potente. Un clúster está desplegado utilizando el sistema gestor de base de datos Microsoft SQL Server versión 10.50. SQL Server soporta transacciones, incluye un entorno gráfico que permite el uso de comandos DDL y DML gráficamente, ofrece la posibilidad de trabajar cliente-servidor y permite administrar información de otros servidores de datos (50).

El otro clúster utiliza el proyecto de código abierto PostgreSQL en su versión 9.4; es un sistema de gestión de bases de datos relacional orientado a objetos y libre, está publicado bajo la licencia PostgreSQL. Algunas de sus características principales son: alta concurrencia, amplia variedad de tipos nativos, uso de desencadenadores, llaves foráneas, entre otras (51).

Como parte de la arquitectura propuesta se propone una capa transversal que contendrá el componente de seguridad WSO2 IS. El Servidor de Identidad de WSO2 es una herramienta de gestión de la seguridad en la comunicación entre los servicios web. Provee gestión de identidades a partir de acceso a servidores LDAP para extraer información de usuarios, así como a través del uso de servicios de *token* de seguridad para su utilización en diversos escenarios de conversación segunda entre servicios. Es una herramienta de fácil utilización para la implementación de los servicios de seguridad de autenticación y autorización. Permite autorización a nivel de código o basada en políticas previamente definidas y almacenadas.

En esta capa se propone además la utilización de un servicio de autenticación centralizada que está desplegado actualmente en la infraestructura de la UIC. El CAS[36] en su versión 4.0.3 permite centralizar las credenciales del usuario para todos los procesos de *login*, permitiendo un inicio de sesión único y persistencia en la autenticación; es decir el usuario se autentica una sola vez y para cambiar de aplicación no es necesario autenticarse nuevamente. Es un proyecto de software libre, con una comunidad muy amplia y de calidad, es muy fácil de instalar y utilizar (52) (53).

Con la definición de los estándares analizados, así como las tecnologías y herramientas que permiten llegar a lo que se conoce como interoperabilidad, se propone la siguiente vista de despliegue de la arquitectura de interoperabilidad de aplicaciones para la UIC.

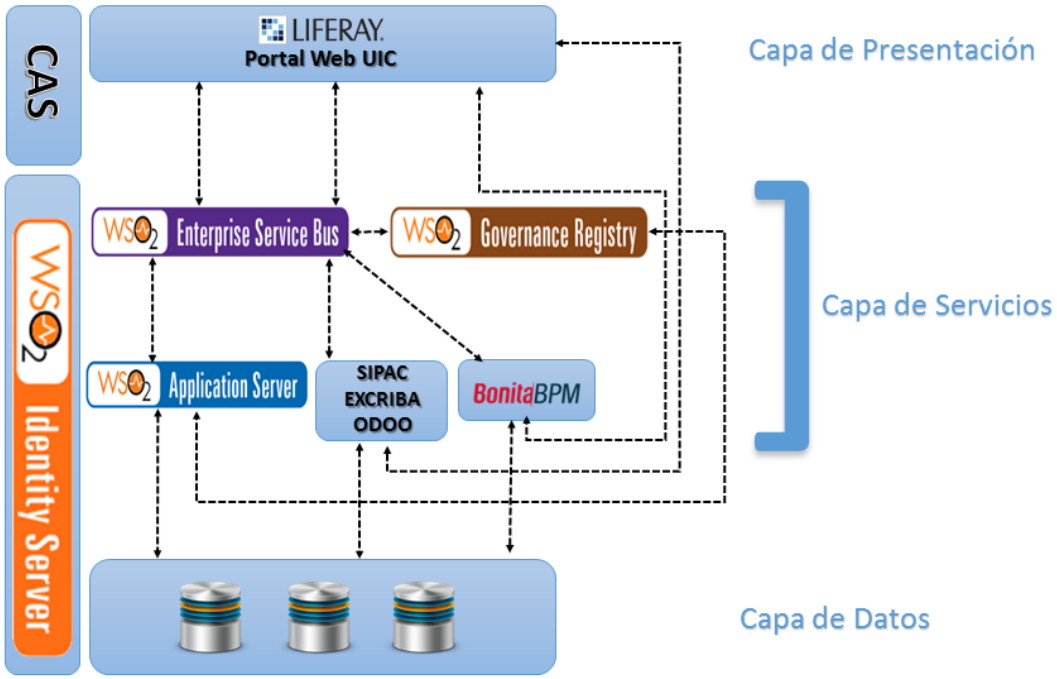

Figura 9. Vista de despliegue de la arquitectura propuesta.

La colaboración que se establece en la arquitectura propuesta parte desde la petición de un usuario en el Portal de la UIC, posteriormente el CAS verifica si el usuario tiene acceso o no a la petición. Una vez enviada la petición a la Capa de Servicios, son necesarios servicios que contengan la

[36] Central Authentication Service

implementación para la ejecución de la petición, para ello, se invoca un servicio que se encuentra alojado en el componente WSO2 ESB o WSO2 AS. Como dicho servicio tiene un formato de mensajes específicos, necesita de un ESB para adecuar el formato, de manera que sea entendible para los demás componentes.

El ESB además, se coordina con el WSO2 GREG para obtener la dirección de localización del servicio web de forma dinámica, lo cual proporciona una flexibilidad a la arquitectura muy importante frente a cambios de localización de los servicios. Como una variante, pudiera establecerse una comunicación con el BonitaBPM, lo que permite utilizar (no gestionar) los procesos creados en el BPM, si la petición necesitara de un proceso de negocio.

De manera transversal, está presente el componente WSO2 IS, que facilita la autenticación y autorización para utilizar el servicio, aunque la petición del usuario haya sido aceptada por el CAS.

A continuación, se describe el RF4 - Consultar miembros:

1. El usuario se autentica en el Portal de la UIC.
2. El CAS verifica que el usuario autenticado tiene los permisos necesarios para consultar las personas que son miembro de la UIC.
3. La petición se envía al WSO2 ESB que es el encargado de localizar el servicio necesario para dicha petición, comunicándose con el componente WSO2 GREG que provee los metadatos del servicio.
4. El ESB utiliza el servicio encontrado en la API de servicios web del eXcriba, en este caso el servicio para la manipulación de contenidos; luego con el método *read()* se obtienen los datos relativos a los nodos de tipo contenido; es decir, se obtienen los datos de todos los miembros, accediendo al repositorio de contenido del eXcriba.
5. El WSO2 IS, verifica si el usuario autenticado tiene acceso al servicio que solicita, y garantiza la integridad de la información.
6. Finalmente, el ESB envía los datos necesarios en un archivo XML, para ser mostrado al usuario en el Portal de la UIC.

Capítulo 2: Descripción de la propuesta de solución

Conclusiones Parciales

En el presente capítulo se describió la propuesta de arquitectura de interoperabilidad para la UIC y los componentes que la conforman. Se detalló el patrón de integración; así como las herramientas y los estándares que se proponen para el desarrollo de la misma y finalmente se expone la propuesta de solución al problema de investigación.

Capítulo 3: Validación de la arquitectura propuesta

El objetivo del presente capítulo es realizar la validación de la arquitectura propuesta, a partir del análisis de los principales métodos y técnicas que permiten evaluar arquitecturas de software. Los mismos se enfocan en el cumplimiento de determinados atributos de calidad, que son evaluados cualitativamente en distintas etapas del proceso de desarrollo de software. A partir del análisis se realizará la selección del método más adecuado para realizar la evaluación.

3.1 Necesidad de evaluar una arquitectura

En el contexto actual, se torna imperativo hablar de calidad en el ámbito de desarrollo de software, ya que la misma garantiza poder competir en el mercado internacional con mayores posibilidades de éxito. Diversas definiciones del término han sido ofrecidas, una de ellas plantea que es un *"proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan*" (7), la ISO/IEC [37] la define como *"la totalidad de rasgos y atributos de un producto de software que le apoyan en su capacidad de satisfacer sus necesidades explícitas o implícitas"*. Sin embargo, los autores del presente trabajo asumen la definición ofrecida por la IEEE, la cual afirma que *"la calidad de un software es el grado en el cual el software posee una combinación deseada de factores"*.

La detección y corrección de errores en etapas tempranas del desarrollo de software es una tarea de suma importancia que puede evitar repercusiones negativas en cuanto a costos y recursos. Realizar una evaluación de la arquitectura de software permite analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad (54).

Generalmente, la evaluación de la arquitectura se realiza después de su especificación y antes de su implementación, en un proceso iterativo y/o incremental que puede realizarse al concluir cada ciclo. Sin embargo, no existe una regla que establezca cuál es el mejor momento, dando flexibilidad para

[37] Intenational Standart Organization

efectuar la evaluación en cualquier etapa del ciclo de vida de una arquitectura. En particular, existen dos etapas: temprana y tardía (55).

Evaluación temprana: no es necesario contar con una arquitectura completamente especificada. Esta puede ser utilizada en cualquier etapa del proceso de creación de la arquitectura, para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes.

Evaluación tardía*:* toma lugar cuando la arquitectura está terminada y la implementación está completa. Este caso ocurre cuando la organización hereda un sistema legado.

Las evaluaciones pueden ser planeadas o no planeadas. La primera es aquella que ha sido planificada por el ciclo de vida de desarrollo, por lo que es parte de las actividades del proyecto. Por el contrario, las no planeadas se presentan cuando la arquitectura contiene varios defectos que han sido detectados en las etapas tardías del desarrollo. Realizar una evaluación no planeada, representa retrasos en los tiempos de entrega, así como un incremento en los costos de un proyecto, por lo que es recomendable que en los proyectos se contemple realizar una o varias evaluaciones a la arquitectura.

La evaluación de la arquitectura permite determinar si la misma es adecuada para el sistema para el cual fue diseñada. Se asume el cumplimiento de esta condición cuando el sistema resultante cumple con los objetivos de calidad y puede ser construido con los recursos disponibles. Este proceso no produce un resultado cuantitativo ya que no resulta de interés, puesto que el sistema no está construido aún. El objetivo que se persigue es aprender cómo un atributo de calidad es afectado por una decisión de diseño arquitectónico, para que de esta manera se pueda estudiar con cuidado dicha decisión.

3.2 Atributos de calidad

Los factores o atributos de calidad, como también se les conoce, representan características o requerimientos adicionales, independientes de los requisitos funcionales, que el sistema debe satisfacer. También pueden ser definidos como las propiedades de un servicio que presta el sistema a sus usuarios (56). Los mismos son la base para la evaluación de una arquitectura, pero por si solos no son suficiente para juzgar la adecuabilidad de la misma, ya que sus descripciones pueden ser ambiguas y estar sujetas a diferentes interpretaciones. Es decir, que los atributos de calidad no son

cantidades absolutas, sino que dependen de un contexto en particular (55). Estos atributos pueden dividirse en dos categorías:

- Observables vía ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.
- No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema.

La descripción de los atributos de calidad se puede consultar en los Anexos 3 y 4.

3.3 Modelos de calidad

Los atributos de calidad pueden organizarse y descomponerse de maneras diferentes, en lo que se conoce como modelos de calidad. La ventaja de los mismos es que permiten definir y medir la calidad, además, contribuyen a comprender las relaciones existentes entre diferentes características de un producto software.

Modelo de McCall

El modelo de McCall, Richards y Walters describe la calidad como un concepto elaborado mediante relaciones jerárquicas entre factores de calidad, en base a criterios y métricas de calidad. Este enfoque es sistemático, y permite cuantificar la calidad a través de las siguientes fases (56):

- Determinación de los factores que influyen sobre la calidad del software.
- Identificación de los criterios para juzgar cada factor.
- Definición de las métricas de los criterios y establecimiento de una función de normalización que define la relación entre las métricas de cada criterio y los factores correspondientes.
- Evaluación de las métricas.
- Correlación de las métricas a un conjunto de guías que cualquier equipo de desarrollo podría seguir.
- Desarrollo de las recomendaciones para la colección de métricas.
- Los factores de calidad se concentran en tres aspectos importantes de un producto de software: características operativas, capacidad de cambios y adaptabilidad a nuevos entornos.

Los factores propuestos por el modelo se describen a continuación (7):

- **Corrección:** grado en el que un programa satisface sus especificaciones y cumple con los objetivos de la misión del cliente.
- **Confiabilidad:** grado en el que se espera que un programa cumpla con su función y con la precisión requerida.
- **Eficiencia:** cantidad de recursos de cómputo y de códigos requeridos por un programa para llevar a cabo su función.
- **Integridad:** grado en el que es posible controlar el acceso de personas no autorizadas al software o a los datos.
- **Usabilidad:** esfuerzo que se requiere para aprender, operar, preparar las entradas e interpretar las salidas de un programa.
- **Facilidad de recibir mantenimiento:** esfuerzo requerido para detectar y corregir un error en un programa.
- **Flexibilidad:** esfuerzo necesario para modificar un programa que ya opera.
- **Susceptibilidad de someterse a pruebas:** esfuerzo que se requiere para probar un programa a fin de garantizar que realiza la función que se pretende.
- **Portabilidad:** esfuerzo que se necesita para transferir el programa de un ambiente de sistema de hardware o software a otro.
- **Reusabilidad:** grado en el que un programa, o sus partes, pueden reutilizarse en otras aplicaciones.
- **Interoperabilidad:** esfuerzo requerido para acoplar un sistema con otro.

Modelo ISO/IEC 9126

Este estándar fue desarrollado para identificar los atributos clave de calidad para un producto de software. Es una simplificación del Modelo de McCall e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software (56). El estándar provee una descomposición de las características en subcaracterísticas, lo cual se muestra en el Anexo 5.

Modelo ISO/IEC 9126 adaptado para arquitecturas de software

Es una adaptación del modelo ISO/IEC 9126 de calidad para efectos de la evaluación de arquitecturas de software (56). A continuación se exponen cómo los requisitos de calidad son

refinados y cómo se adaptan a la arquitectura de software, teniendo en cuenta que las características y sub-características se consideran independientes (57).

Funcionalidad

Adecuación. Esta sub-característica se basa en que el producto posea las funciones necesarias para las tareas que debe desempeñar.

A nivel arquitectónico:

- En este caso se identifican todas las funcionalidades del sistema y cada una de ellas se refina en un atributo cuyo valor es sí (1) o no (0). Pueden definirse atributos que tengan como rango de valor [1…0], expresando sentido de presencia o ausencia del atributo. La métrica en este caso constituye una escala para obtener niveles de calificación.
- La secuencia de los diagramas obtenidos a partir de los requisitos funcionales se refina. En caso de haber una especificación de la arquitectura, la funcionalidad especificada se descompone en funciones asociadas a los componentes y esta composición se verifica contra los requerimientos del sistema.

Exactitud. Proporciona que los resultados sean los correctos o los esperados con un grado necesario de precisión. Puede ser medida por un atributo que verifique a través del código fuente o a través de la medición de las funciones de cada componente.

A nivel arquitectónico:

- Se identifican los componentes responsables de funciones vinculadas directamente a los cálculos de precisión.
- El atributo se calcula mediante las siguientes mediciones: Πi (Exactitud (componentes funcionales **i**))

Interoperabilidad. Evidencia la capacidad del producto de establecer una interacción entre sus componentes y/o con otros sistemas externos.

***A nivel arquitectónico*:**

- Identificar los conectores de comunicación con sistemas externos.
- Es refinada en un atributo cuyo valor es sí o no.

Seguridad. La capacidad para prevenir el acceso no autorizado a programas o datos.

A nivel arquitectónico:

- Significa contar con un mecanismo o dispositivo (software o hardware) para llevar a cabo esta tarea de forma explícita. Puede ser un componente (por ejemplo, un servicio proporcionado por el *middleware*) o una funcionalidad integrada en un componente.
- Es refinada en un atributo cuyo valor es sí o no, dependiendo de la presencia o no del mecanismo o dispositivo.

Confiabilidad

Madurez. Evidencia la capacidad del producto de software para evitar fallas, como resultado de las fallas en el software. Es refinada en un atributo *Mean Time To Failure* (MTTF) medido en el código fuente.

A nivel arquitectónico:

- El atributo se calcula mediante las siguientes mediciones: Σi madurez (componente I) + Σj madurez (conector j).

Tolerancia a fallos. Refleja la capacidad de mantener un determinado nivel de rendimiento en caso de fallos de software o de vulnerabilidad en su interfaz especificada.

A nivel arquitectónico:

- Significa contar con un mecanismo o dispositivo de software. Puede ser un componente independiente o una funcionalidad integrada en un componente, por ejemplo el manejo de excepciones o redundancia.
- Es refinada en un atributo cuyo valor es sí o no, dependiendo de la presencia o no del mecanismo o dispositivo.
- Puede ser refinado en un atributo cuyo valor se asocia con el mecanismo o dispositivo.

Recuperación. Esta sub-característica se expresa en tres elementos importantes, la capacidad para restablecer el nivel de rendimiento, la capacidad de recuperar los datos y el tiempo y el esfuerzo necesario para ello.

A nivel arquitectónico:

- Esto significa la existencia de un mecanismo o dispositivo de software, que puede ser un componente o una parte de un componente, que permita restablecer el nivel de rendimiento y la recuperación de los datos, por ejemplo, la redundancia.
- Si el mecanismo existe, la recuperación se refina en el cálculo de los parámetros de los atributos relacionados con tiempo y esfuerzo. Debe ser calculado para cada componente, mecanismo o dispositivo.

Eficiencia

Rendimiento. Refleja la capacidad del producto de software para proporcionar el tiempo de respuesta adecuado, tiempo de transformación y las tasas de rendimiento en el desempeño de su función bajo condiciones establecidas. Es un atributo que puede ser medido para cada funcionalidad del sistema.

A nivel arquitectónico:

- Se mide para cada función y cada funcionalidad que ejecute el usuario por medio de atributos calculados por las siguientes mediciones: Σ i Performance (funcionalidad de los componentes i) + Σ j Performance (conector j).

Aprovechamiento de los recursos. Manifiesta la cantidad y tipos de recursos utilizados en el desempeño de una función, así como la duración de esos usos.

A nivel arquitectónico:

- Los atributos pueden ser definidos y medidos para cada función, asociados al estilo arquitectónico utilizado. Espacio y tiempo están asociados a los componentes.

Mantenibilidad

Analizabilidad. Mide la capacidad del producto de software de diagnosticar las deficiencias, posibles causas de fallos, o para identificar las partes que deben ser modificadas.

Facilidad para el cambio. Capacidad del producto de software para permitir una modificación específica que debe aplicarse.

Estabilidad. Capacidad del producto para evitar los efectos inesperados de las modificaciones del software, en especial el riesgo que provocan estas modificaciones.

Testeabilidad. Expresa la capacidad que presenta el producto de medir el esfuerzo necesario para validar el software modificado. Esta sub-característica es refinada en un atributo que mide la complejidad del código fuente, a través del cálculo de métricas de tamaño.

A nivel arquitectónico:

- Se tiene en cuenta la sub-característica de acoplamiento, que es una propiedad global de la arquitectura en relación con los intercambios entre los componentes.
- Otra sub-característica es la modularidad, que expresa la topología de la arquitectura, como el número de componentes en función de un componente. Se trata de un atributo en el que se calcula la participación de cada componente, mediante métricas de tamaño.

Portabilidad

Adaptabilidad. Refleja la capacidad del producto de software para adaptarse a diferentes entornos especificados utilizando sólo su propia funcionalidad.

A nivel arquitectónico:

- La presencia de mecanismos de adaptación.
- Esta sub-característica es refinada en un atributo cuyo valor es sí o no, dependiendo de la presencia o no del mecanismo.

Capacidad de instalación. Permite medir la capacidad del producto de software para ser instalado en un entorno determinado.

A nivel arquitectónico:

- La presencia de un mecanismo de instalación.
- Esta sub-característica es refinada en un atributo cuyo valor es sí o no, dependiendo de la presencia o no del mecanismo.

Coexistencia. Refleja la capacidad del producto de software para coexistir con otros programas informáticos independientes en un entorno común, compartiendo recursos comunes.

A nivel arquitectónico:

- La presencia de un mecanismo que facilite la coexistencia entre sistemas.
- Esta sub-característica es refinada en un atributo cuyo valor es sí o no, dependiendo de la presencia o no del mecanismo.

Reemplazo. Mide la capacidad del producto de software de reemplazar otro producto de software determinado para el mismo propósito en el mismo entorno. Se mide fundamentalmente la adaptabilidad y la capacidad de instalación.

A nivel arquitectónico:

- El atributo se expresa en una lista de los componentes reemplazables, para cada componente.

Selección del modelo de calidad a emplear

El modelo McCall a pesar de realizar un refinamiento de los factores de calidad que propone, posee métricas de carácter general, muy difíciles de aplicar a la arquitectura de software como producto. Por su parte, la adecuación del estándar ISO 9126 para el caso específico de la arquitectura de software como producto intermedio del ciclo de desarrollo, es el modelo que más elementos aporta al análisis de la calidad arquitectónica. No solo se analizan cada una de las seis características que propone la norma ISO/IEC 9126-1 y sus sub-características desde un punto de vista arquitectónico, sino que facilita la comprensión acerca de cuáles elementos debe tener una arquitectura de software para lograr valores adecuados de cada atributo de calidad.

Los resultados de la aplicación del modelo seleccionado se pueden consultar en el Anexo 6.

3.4 Técnicas de evaluación

A pesar de que los modelos de calidad logren medir el grado de cumplimiento de los factores de calidad en un producto específico, la evaluación de la arquitectura de software no abarca solamente ese marco de medición. Evaluar una arquitectura implica además, llevar a cabo un proceso de priorización de características de calidad y valorar mediante escenarios que elemento arquitectónico debe ser cambiado para mejorar atributos de rendimiento o mantenibilidad (57). Por estas razones, a continuación se analizan un conjunto de técnicas y métodos que separan el concepto de calidad de la evaluación arquitectónica y va más enfocado a la arquitectura de software como disciplina.

Capítulo 3: Validación de la arquitectura propuesta

Con el objetivo de tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos. En este sentido, existen un grupo de técnicas para la evaluación de la calidad, las cuales se dividen en dos grupos: cualitativas y cuantitativas. Por lo general, las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción, mientras que las cuantitativas, se usan cuando la arquitectura ya ha sido implantada. El Anexo 7 muestra con más detalle la clasificación de estas técnicas (54).

Debido a las características de la propuesta de solución y al alto costo que implica la aplicación de técnicas cuantitativas, solo se hará énfasis durante la presente investigación en las técnicas de evaluación cualitativas basadas en escenarios.

Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. El mismo consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc. El contexto describe qué sucede en el sistema al momento del estímulo y la respuesta describe a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado (56).

Los escenarios permiten concretar y comprender atributos de calidad. Su importancia radica en que son fáciles de crear y entender, son poco costosos, no requieren mucho entrenamiento y son muy efectivos. Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Árbol de utilidad (*Utility Tree*) y los Perfiles (*Profiles*).

Árbol de utilidad

Es un esquema en forma de árbol que presenta los atributos de calidad de un sistema, refinados hasta el establecimiento de escenarios que especifican detalladamente el nivel de prioridad de cada uno. La intención de su uso es la identificación de los atributos de calidad más importantes para un proyecto en particular. Tiene como característica que no existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la

construcción del árbol. El mismo tiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol, los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado (56).

Perfiles

Un perfil representa un conjunto de escenarios con alguna importancia relativa asociada a ellos. Su uso permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Este instrumento de evaluación tiene asociado dos formas de especificación: perfiles completos y perfiles seleccionados.

Los primeros definen todos los escenarios relevantes como parte del perfil, lo cual permite realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, incluyendo todos los posibles casos. Generalmente, su uso se reduce a sistemas relativamente pequeños y sólo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad. Los segundos se asemejan al trabajo con muestras poblacionales en estadística, tomando un conjunto de escenarios de forma aleatoria, de acuerdo a determinados requerimientos.

3.5 Métodos de evaluación de arquitecturas de software

Existen varios métodos en la actualidad que permiten la realización de evaluaciones arquitectónicas, cada uno de ellos enfocados en distintas técnicas de evaluación y diversos aspectos y atributos de calidad. Obtener una evaluación más eficiente depende de las condiciones y los atributos definidos.

SAAM (*Software Architecture Analysis Method*)

Fue el primer método de evaluación basado en escenarios que surgió y se centra sobre todo en la modificabilidad. En la práctica, ha demostrado ser útil para evaluar muchos atributos de calidad rápidamente, como portabilidad, modificabilidad, extensibilidad, integrabilidad, así como el cubrimiento funcional que tiene la arquitectura sobre los requerimientos del sistema.

SAAM puede ser utilizado para evaluar una o múltiples arquitecturas. Si se comparan dos o más se culmina el análisis con una tabla indicando las fortalezas y debilidades de cada una en cada escenario. Si se evalúa una sola, se culmina con un reporte señalando los componentes

computacionales donde la arquitectura no alcanza el nivel requerido. En ningún caso se emite un valor absoluto acerca de la calidad arquitectónica.

El método de evaluación de SAAM consta de seis pasos que se describen a continuación:

1. Desarrollo de escenarios: el objetivo fundamental es capturar las principales actividades que el sistema debe soportar. Los escenarios encontrados deben reflejar los atributos de calidad de interés y también mostrar la interacción entre las diferentes personas que utilizarán el sistema. Los mismos deben:
    - Reflejar el punto de vista de cada interesado.
    - Reflejar los atributos más importantes requeridos.
    - Reflejar la variedad de interacciones posibles.
    - El total de escenarios debe ser un número manejable, es decir, deben existir los recursos suficientes como para desarrollarlos, ejecutarlos y evaluarlos.
2. Descripción de la arquitectura: se presentan las arquitecturas candidatas. Es fundamental que la notación empleada sea bien entendida por todos los involucrados y que los elementos presentados sean comunes para todas las arquitecturas candidatas.
3. Clasificación de escenarios: se deben clasificar los escenarios como directos o indirectos.
4. Evaluación de escenarios: para cada escenario indirecto, se deben listar los cambios necesarios en la arquitectura para soportarlo, y el costo de llevarlos a cabo debe ser estimado.
5. Interacción de escenarios: deben determinarse las interacciones de escenarios sobre cada componente de cada arquitectura.
6. Evaluación general: se asigna un peso a cada escenario en términos de su influencia para que el sistema sea exitoso. El peso puede ser elegido de acuerdo a los objetivos del negocio, costos, riesgos, etc.

SAAM es un método para la evaluación temprana de arquitecturas, con respecto a algunos atributos de calidad expresados en el contexto de circunstancias específicas (escenarios). Proporciona un marco conceptual suficiente para saber qué es necesario, pero no suficiente para realizar una evaluación completa de la arquitectura.

ATAM (*Architecture Tradeoff Analysis Method*)

Determina cuan bien una arquitectura particular satisface las metas de calidad, provee ideas de cómo esas metas de calidad interactúan y realizan concesiones entre ellas. El ATAM también puede ser utilizado para analizar sistemas legados ya que incrementa el entendimiento de los atributos de calidad en dichos sistemas (55).

La parte principal del método consta de nueve pasos, que a su vez se dividen en cuatro grupos:

- Presentación: donde la información es intercambiada. Se describe el método, se presentan las pautas del negocio y la arquitectura.
- Investigación y análisis: se valoran los atributos claves de calidad requeridos. Se identifican y analizan las propuestas arquitectónicas y se genera el árbol de utilidad de los atributos de calidad.
- Pruebas: se revisan los resultados obtenidos contra las necesidades relevantes de los *stakeholders.*
- Informes: se presentan los resultados del ATAM.

Método de Losavio

La especificación de los atributos de calidad haciendo uso de un modelo basado en estándares internacionales ofrece una vista amplia y global de los atributos de calidad, tanto a usuarios como arquitectos del sistema, para efectos de la evaluación. El método contempla siete actividades, las mismas son:

1. Analizar los principales requerimientos funcionales y no funcionales del sistema, para establecer las metas de calidad.
2. Utilizar el modelo de calidad ISO/IEC 9126 adaptado para arquitecturas de software. Algunas métricas pueden definirse con mayor nivel de detalle.
3. Presentar las arquitecturas candidatas iniciales.
4. Construir la tabla comparativa para las arquitecturas candidatas.
5. Establecer prioridades para las subcaracterísticas de calidad tomando en cuenta los requerimientos de calidad del sistema.
6. Analizar los resultados obtenidos y resumidos en la tabla, de acuerdo con las prioridades establecidas.

Selección del método de evaluación a emplear

Capítulo 3: Validación de la arquitectura propuesta

A partir del análisis de los modelos de calidad expuestos, se selecciona para la evaluación de la arquitectura de interoperabilidad propuesta en el presente trabajo de diploma el método de Losavio. La elección del mismo se fundamenta en que permite evaluar la arquitectura en una etapa posterior a la culminación del diseño y anterior a la implementación, garantizando que pueda observarse el cumplimiento de los atributos de calidad asociados al sistema, así como cuál será su comportamiento general. Además, los atributos de calidad en los que se basa se relacionan directamente con la arquitectura.

3.6 Resultados de la evaluación

Tabla 1. Resultados de la aplicación del método de Losavio

Características	Subcaracterísticas	Escenario
Funcionalidad	Adecuación	Los componentes del sistema cumplen la función para la cual serán utilizados.
	Interoperabilidad	El sistema posee componentes capaces de leer datos provenientes de otros sistemas y de producir datos para otros sistemas.
	Seguridad	El sistema detecta la actuación de un intruso e impide el acceso a los componentes que manejan información sensible. Asegura que los componentes no pierdan datos ante un ataque interno o externo.
Confiabilidad	Tolerancia a fallos	Todas las operaciones ejecutadas por los componentes se realizan correctamente bajo condiciones adversas.
	Recuperabilidad	Los componentes del sistema no fallan bajo ciertas condiciones especificadas. Ante problemas con el ambiente un subconjunto determinado de los componentes puede continuar prestando sus servicios.
	Facilidad para el cambio	El sistema facilita la sustitución/adaptación de un componente.
	Estabilidad	El sistema brinda facilidad para adaptar un componente.
Portabilidad	Adaptabilidad	El sistema debe continuar funcionando correctamente aun cuando los servicios de los componentes provistos por el ambiente varíen.
	Coexistencia	Los componentes manejan adecuadamente recursos compartidos del sistema.

	Reemplazabilidad	Los componentes pueden ser reemplazados por otros que cumplan con los mismos estándares.

Conclusiones Parciales

En el presente capítulo se expusieron varias de las técnicas utilizadas para evaluar una arquitectura. Se definieron los atributos de calidad a evaluar y se realizó la evaluación del diseño de la arquitectura de interoperabilidad de aplicaciones para la UIC de manera parcial, demostrando que la misma condiciona los atributos de calidad deseados.

Conclusiones

- El estudio del estado del arte permitió definir que la alternativa más adecuada para realizar la propuesta de solución era la realización del diseño de una arquitectura orientada a servicios, tomando como base para la misma el empleo de servicios web.
- El estudio de los estándares y herramientas que se emplean para el desarrollo de arquitecturas de interoperabilidad, permitió seleccionar los más adecuados para emplearlos en la arquitectura propuesta.
- Se diseñó una arquitectura basada en tres capas que permite la interoperabilidad entre aplicaciones heterogéneas.
- El análisis de los métodos y técnicas de evaluación de arquitecturas de software y la aplicación del método de Losavio garantizó que la arquitectura propuesta tiene la calidad requerida.

Recomendaciones

Al concluir la presente investigación se proponen, una serie de tareas para ampliar y dar continuidad al trabajo realizado:

- Se recomienda adoptar el ciclo de vida propuesto con el objetivo de realizar una arquitectura organizada desde su principio, analizando las particularidades y requerimientos de la propuesta.
- Investigar el resto de los productos que integran la suite WSO2 y analizar la inclusión de otros componentes que aporten nuevas funcionalidades a la arquitectura.

Referencias bibliográficas

1. *Propuestas de Estatutos de la Organización.pdf* [online]. [Accessed 3 November 2015]. Available from: http://www.mincom.gob.cu/sites/default/files/Propuestas%20de%20Estatutos%20de%20la%20Organizaci%C3%B3n.pdf

2. AZÁN BASALLO, Yasser, BASALLO, Yasser Azán and ESTRADA, Anay Díaz. Una experiencia sobre integración de aplicaciones informáticas. *Serie Científica* [online]. 21 April 2010. Vol. 3, no. 4. Available from: https://publicaciones.uci.cu/index.php/SC/article/view/216

3. PALACIOS, Ruiz and ANTONIO, Marco. Especificación y desarrollo de un sistema basado en tecnologías semánticas para la integración e interoperabilidad de aplicaciones heterogéneas. [online]. 2013. [Accessed 6 June 2016]. Available from: http://e-archivo.uc3m.es/handle/10016/17159

4. MARTÍNEZ USERO, José Angel and PALACIOS-RAMOS, Elsa. XML: un medio para fomentar la interoperabilidad, explotación y difusión de contenidos en la administración electrónica. In : [online]. 2003. [Accessed 28 March 2016]. Available from: http://eprints.rclis.org/4292/

5. CORREA, Alexsandra, DI PETTA, Estefany and FRUGONI, Valentina. Interoperabilidad en los sistemas de información. [online]. 2011. [Accessed 28 March 2016]. Available from: https://www.colibri.udelar.edu.uy/handle/123456789/432

6. BASS, L, CLEMENTS. P and KAZMAN, R. *Software Architecture in Pratice*. 2003.

7. Roger S. Pressman. *Ingeniería del Software. Un enfoque práctico*. 7. McGraw Hill, 2010.

8. MORENO ESCOBAR and HERNAN. *Libro blanco de interoperabilidad de gobierno electrónico para América Latina y el Caribe*. 2007.

9. Hernán Moreno Escobar, HUGO SIN TRIANA and NETTO, Sérgio Caino Silveira. *Conceptualización de arquitectura de gobierno electrónico y plataforma de interoperabilidad para América Latina y el Caribe* [online]. July 2007. Available from: http://www.cepal.org/SocInfo

10. HOHPE, G. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. 2003. ISBN 978-0-321-20068-6.

11. RAHUL SHARMA, BETH STEARNS and TONY NG. *J2EE Connector Architecture and Enterprise Application Integration*. Addison – Wesley, 2003. ISBN 0-201-77580-8.

12. DUVALL P. S and ANDREW M. *Continuous Integration, Improving Software Quality and Reducing Risk* [online]. 2007. Addison-Wesley. Available from: http://www.addisonwesley.de/main/main.asp?page=englisch/bookdetails&ProductI D=121548

13. MANGARELLI EDUARDO and CABRERA MARTÍN. *SOA Introduction*. 2006.

14. G. LÓPEZ, A. ECHEVERRÍA, P. FIERRO and I. JEDER. *UNA PROPUESTA DE MODELOS DE CICLO DE VIDA (MCVS) PARA LA INTEGRACIÓN DE LOS PROCESOS DE NEGOCIO UTILIZANDO SERVICE ORIENTED ARCHITECTURE (SOA)*.

15. LUÍS FERNANDO GONZÑALEZ ALVARÁN, ADRIANA XIOMARA REYES GAMBOA and GLADIS HELENA VÁSQUEZ ECHAVARRÍA. *DISEÑO DE APLICACIONES WEB BASADAS EN ARQUITECTURAS ORIENTADAS A SERVICIOS (AOS), UTILIZANDO WEBML*.

16. SIMOES DORIVAL. *El paso que sigue* [online]. December 2007. Available from: http://64.116.224.233/detras_el_paso_que_sigue/fichas/soa.html

17. MARIANELA DIAZ ROSALES. Arquitectura orientada a servicios. In : *Convención Científica de Ingeniería y Arquitectura*. December 2008.

18. PABLO PERIS SOLER and NOEMÍ BELENGUER NAVARRO. *Servicios Web*.

19. DPTO DE CIENCIAS E INTELIGENCIA ARTIFICIAL. UNIVERSIDAD DE ALICANTE. *Introducción a los Servicios Web. Invocación de servicios web SOAP.* [online]. June 2014. Available from: http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes

20. OTÓN TORTOSA, Salvador. Propuesta de una arquitectura software basada en servicios para la implementación de repositorios de objetos de aprendizaje distribuidos. [online]. 2006. [Accessed 29 March 2016]. Available from: http://dspace.uah.es/dspace/handle/10017/472

21. MAURICIO ROJAS C (ÚLTIMO) and MONTILVA, Jonás. *Una arquitectura de software para la integración de objetos de aprendizaje basada en servicios web*. August 2011.

22. MARIO VEGA-BARBAS (ÚLTIMO), MANSILLA, Diego Casado and JUAN R. VELASCO. *S3OiA: Propuesta de Arquitectura para la Interoperabilidad en la Internet de las Cosas* [online]. September 2011. [Accessed 5 November 2015]. Available from: http://www.researchgate.net/publication/225090312

23. CASTRILLÓN, Helder Y., CAROLINA GONZÁLEZ and DIEGO M. LÓPEZ. *Modelo Arquitectónico para Interoperabilidad entre Instituciones Prestadoras de Salud en Colombia*. July 2012.

24. MATOS, Madelaine Hernández and ONIER HOGGUIT RODRÍGUEZ. *Propuesta de arquitectura de interoperabilidad para el intercambio de datos en el Órgano de Justicia-MININT*. June 2011.

25. BASTARRICA MARIA CECILIA, GUTIÉRREZ CLAUDIO and OCHOA SERGIO. *Documentación electrónica e interoperabilidad de la información* [online]. 1ra. Universidad de Chile : Maval Ltda, 2009. ISBN 978-956-19-0633-4. Available from: www.dcc.uchile.cl/librointeroperabilidad

26. MONTERO AYALA, RAMÓN. *XML Iniciación y referencia*. 1ra. 2001.

27. *XML JSON YAML formatos para intercambiar información* [online]. May 2014. Available from: http://hipertextual.com/archivo/2014/05/xml-json-yaml/

28. UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS. *Arquitectura para los sistemas que conforman la Intranet Universitaria*. 2007.

29. NARANJO, MAURICIO. *Lineamientos para adopcion de arquitectura orientada a servicios para las empresas*. Bogotá D.C, 2007.

30. REST vs SOAP al servicio de la web. *INUSUAL* [online]. 31 January 2014. [Accessed 30 March 2016]. Available from: http://inusual.com/articulos/rest-vs-soap-al-servicio-de-la-web/

31. MARTÍNEZ, PEDRO ESPINA. *Extensibilidad de UDDI*. 2007.

32. [SOA] – Introducción A UDDI | El Club Del Programador. [online]. [Accessed 30 March 2016]. Available from: http://www.elclubdelprogramador.com/2011/10/03/soa-introduccion-a-uddi/

33. XML Encryption Syntax and Processing Version 1.1. [online]. [Accessed 25 May 2016]. Available from: https://www.w3.org/TR/xmlenc-core1/

34. XML Signature Syntax and Processing (Second Edition). [online]. [Accessed 25 May 2016]. Available from: https://www.w3.org/TR/xmldsig-core/

35. XML.com. [online]. [Accessed 25 May 2016]. Available from: http://www.xml.com/pub/a/2001/08/08/xmldsig.html

36. What is WS-Security (Web Services Security)? - Definition from WhatIs.com. [online]. [Accessed 25 May 2016]. Available from: http://searchsoa.techtarget.com/definition/WS-Security

37. *Introducción a SAML* [online]. Available from: http://bibing.us.es/proyectos/abreproy/11214/fichero/TOMO+I%252F08+Capitulo+8+Introduccion+a+SAML.pdf

38. *El estándar SAML* [online]. Available from: http://bibing.us.es/proyectos/abreproy/11147/fichero/Memoria%252F04+-+El+est%E1ndar+SAML.pdf

39. What is an ESB? *MuleSoft* [online]. 21 May 2015. [Accessed 25 May 2016]. Available from: https://www.mulesoft.com/resources/esb/what-esb

40. ¿Qué es un BPM? Business Process Management. [online]. [Accessed 25 May 2016]. Available from: https://www.auraportal.com:443/es/-que-es-un-bpm--business-process-management

41. FERNÁNDEZ ARTURO. *Portales empresariales con Liferay.* 2006.

42. WSO2. *WSO2*. 2015.

43. WSO2. *WSO2 Enterprise Service Bus*. 2015.

44. WSO2. *WSO2 Application Server*. 2015.

45. Governance Registry | WSO2 Inc. [online]. [Accessed 25 May 2016]. Available from: http://wso2.com/products/governance-registry/

46. Sobre alfresco - la alternativa para la gestión de contenidos empresariales de código libre. [online]. February 2012. Available from: http://www.alfresco.com/es/about/

47. Alfresco Content Management Web Services - alfrescowiki. [online]. [Accessed 3 June 2016]. Available from: https://wiki.alfresco.com/wiki/Alfresco_Content_Management_Web_Services

48. *Manual de usuario de planificación*. September 2015.

49. Portal OpenErp (Odoo) en España. *Odoo - OpenERP - ERP, CRM, MRP, SGA 100% Libre - | Sin Licencias* [online]. 21 May 2013. [Accessed 3 June 2016]. Available from: http://openerpspain.com/

50. Cluster sql server. [online]. 15:10:20 UTC. [Accessed 25 May 2016]. Available from: http://es.slideshare.net/orellana_22/cluster-sql-server

51. www.postgresql.org.es. [online]. [Accessed 25 May 2016]. Available from: http://www.postgresql.org.es/

52. *Microsoft PowerPoint - Presentación Estudio_CAS.pptx - 2_Estudio_CAS.pdf* [online]. [Accessed 25 May 2016]. Available from: http://www.si.ulpgc.es/sites/default/files/images/stories/SIC/Jornadas/Jornada11/2_Estudio_CAS.pdf

53. LDAP-SSO - Documentación - Facultad de Psicología. [online]. [Accessed 25 May 2016]. Available from: http://projek.psico.edu.uy/projects/doc-ui/wiki/LDAP-SSO

54. Procedimiento para la evaluación de arquitecturas de software basadas en componentes. *CuriositySec* [online]. 4 June 2014. [Accessed 4 May 2016]. Available from: http://curiositysec.com/procedimiento-para-la-evaluacion-de-arquitecturas-de-software-basadas-en-componentes/

55. MAURICIO DÁVILA, MARTÍN GERMÁN, DIEGO CRUTAS and ANDRÉS GARCÍA. *Evaluación de arquitecturas de software*.

56. ERIKA CAMACHO, FABIO CARDESO and GABRIEL NUÑEZ. *Arquitecturas de software*. April 2004.

57. ING. YAMILA VIGIL REGALADO. *Metodología de evaluación de arquitecturas de software*. Maestría. Universidad de las Ciencias Informáticas, 2009.

Anexos

Anexo 1 Tipos de datos básicos en UDDI

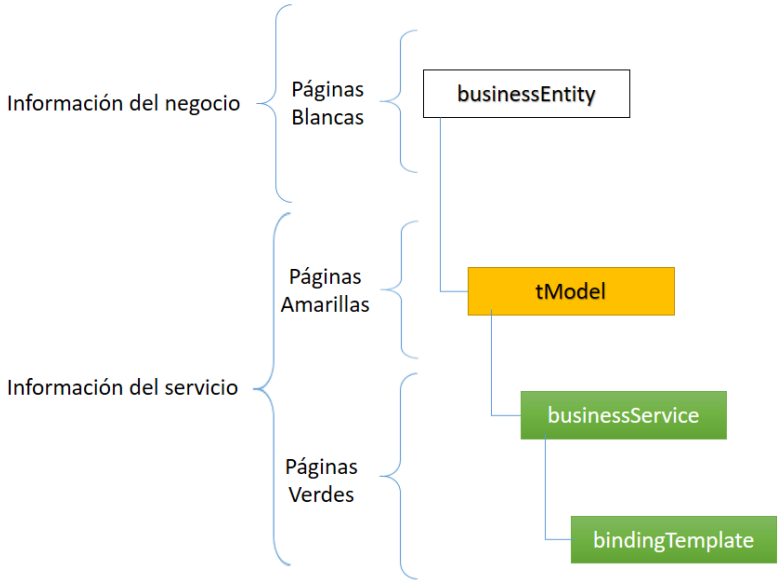

Figura 7. Tipos de datos básicos en UDDI.

Anexo 2 Detalles de los datos UDDI

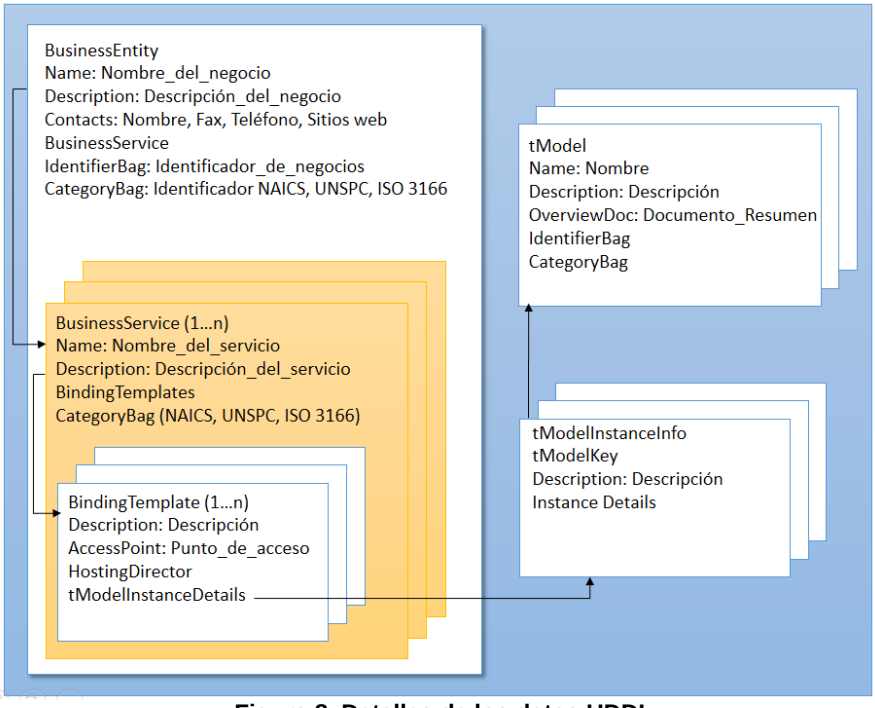

Figura 8. Detalles de los datos UDDI.

Anexo 3 Atributos de calidad observables vía ejecución

Tabla 2. Descripción de atributos de calidad observables vía ejecución.

Atributo de Calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso.
Confidencialidad	Es la ausencia de acceso no autorizado a la información.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
Confiabilidad	Es la medida de la habilidad de un sistema para mantenerse operativo a lo largo del tiempo.
Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Anexo 4 Atributos de calidad no observables vía ejecución

Tabla 3. Descripción de atributos de calidad no observables vía ejecución.

Atributo de Calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto de realizar ciertos cambios al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información.
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.

Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes.
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, datos o procedimental.
Capacidad de Prueba	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas.

Anexo 5 Características y subcaracterísiticas de calidad

Tabla 4. Características y subcaractersticas de calidad. (Pressman, 2002)

Característica	Subcaracterística
Funcionalidad	Adecuación, Exactitud, Interoperabilidad, Seguridad
Confiabilidad	Madurez, Tolerancia a fallas, Recuperabilidad
Usabilidad	Entendibilidad, Capacidad de aprendizaje, Operabilidad
Eficiencia	Comportamiento en tiempo, Comportamiento de recursos
Mantenibilidad	Analizabilidad, Modificabilidad, Estabilidad, Capacidad de pruebas
Portabilidad	Adaptabilidad, Instabilidad, Reemplazibilidad

Anexo 6 Resultados de la aplicación del modelo de calidad

Tabla 5. Resultados de la aplicación del modelo de calidad.

Característica	Subcaracterística	Arquitectura en 3 capas	Comentarios y resultados
Funcionalidad	Adecuación	si	
	Exactitud	-	Depende del código fuente.
	Interoperabilidad	si	
	Seguridad	si	

Confiabilidad	Madurez	-	Depende del código fuente.
	Tolerancia a fallos	si	
	Recuperación	si	
Mantenibilidad	Analizabilidad	-	
	Facilidad para el cambio	si	
	Estabilidad	si	
	Testeabilidad	-	Depende del código fuente.
Portabilidad	Adaptabilidad	si	
	Capacidad de instalación	-	
	Coexistencia	si	
	Reemplazo	si	

Anexo 7 Clasificación de las técnicas de evaluación

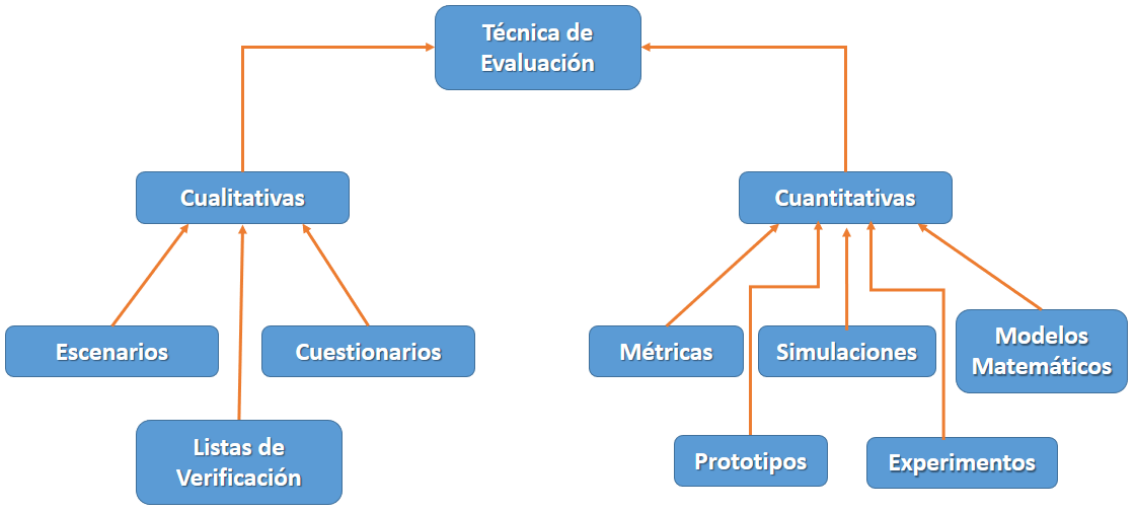

Figura 10. Clasificación de las técnicas de evaluación.

Anexo 8 Comparación entre los métodos de evaluación

Tabla 6. Comparación de los métodos de evaluación.

	ATAM	SAAM	Losavio
Atributos de calidad	• Modificabilidad • Seguridad	• Modificabilidad • Funcionalidad	• Funcionalidad • Confiabilidad

contemplados	• Confiabilidad • Desempeño		• Usabilidad • Eficiencia Mantenibilidad • Portabilidad
Objetos analizados	• Estilos arquitectónicos • Documentación • Flujos de datos • Vistas arquitectónicas	• Documentación • Vistas Arquitectónicas	• Especificación de atributos de calidad
Etapas del proyecto en las que se aplica	Luego de que el diseño de la arquitectura ha sido establecido.	Luego de que la arquitectura cuenta con funcionalidad ubicada en módulos.	Luego de que el diseño de la arquitectura ha sido establecido.
Enfoques utilizados	• Árbol de utilidad y lluvia de ideas para articular los requerimientos de calidad. • Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos.	• Lluvia de ideas para escenarios y articular los requerimientos de calidad. • Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios.	Análisis y comparación de los resultados obtenidos para las arquitecturas candidatas.