Universidad de las Ciencias Informáticas.

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

Título: Propuesta de solución a la Vista de Arquitectura de Integración del Sistema GESPRO 12.05.

Autor: Pablo Enrique Noriega Ordax.

Tutores: Ing. Yordanis Rodríguez Rodríguez.

Ing. Yenisleidy Piloto Lastra.

Ciudad de La Habana, Junio 2012.

Declaración Jurada de Autoría

Declaro ser el autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmo a los ___días del mes de _____del año 2012.

Pablo Enrique Noriega Ordax

_____

Firma del Autor

Ing. Yordanis Rodríguez Rodríguez

_____

Firma del Tutor

Ing. Yenisleidy Piloto Lastra

_____

Firma del Tutor

RESUMEN

La Universidad de las Ciencias Informáticas (UCI), se encuentra inmersa en la realización de un Sistema Integral para la gestión de proyectos (GESPRO) puesto que las herramientas con que cuenta para este objetivo no cubren las expectativas. Este sistema de Gestión no posee una descripción arquitectónica, por lo que surge la necesidad de realizar la vista de arquitectura de integración.

En el presente trabajo se realiza una propuesta para la vista de arquitectura de integración del sistema GESPRO, para lograr su correcta interoperabilidad. Se realiza un estudio del estado del arte sobre el tema enmarcado en el campo de acción.

Al mismo tiempo, se realiza el análisis sobre los elementos a tener en cuenta para la evaluación de la arquitectura, de ellos, se utiliza la técnica basada en escenarios con el instrumento Árbol de Utilidades y el método de evaluación ATAM, con el objetivo de identificar riesgos y fortalezas además de comprobar que el sistema cumple con los atributos de calidad seleccionados.

Palabras Claves: arquitectura, gestión de proyectos, integración, interoperabilidad.

ÍNDICE

Índice de tablas

Índice de figuras

INTRODUCCIÓN

En el mundo actual, las Tecnologías de la Información y las Comunicaciones (TIC), han revolucionado el planeta, dando lugar al surgimiento de la era de la información. Las TIC son consideradas la base para construir una Sociedad de la Información y del Conocimiento que permita generar nuevas oportunidades de acceso a la información e impulsar el desarrollo. Por lo que los cambios tecnológicos en el campo de la informática hacen que esta ciencia sea cada día más eficiente.

En la actualidad, el desarrollo informático está marcado por la utilización de distintas tecnologías y arquitecturas incompatibles entre sí, lo que imposibilita compartir recursos y contenidos e integrar diversos sistemas. Gracias a la interoperabilidad, sistemas concebidos para realizar distintas funciones, pueden comunicarse. No se conciben aplicaciones no conectadas; integración es progreso, disminución de esfuerzo, reutilización y futuro, sin embargo, aunque existen diversas alternativas para la integración, aún no se logra del todo, es común que todavía existan problemas de disponibilidad, de redundancia de la información, debido a varios factores entre los que se encuentran las limitaciones económicas y tecnológicas.

Muchas organizaciones en el mundo se encuentran orientadas a proyectos, por lo que necesitan manejar la información mediante sistemas para la gestión de los mismos. Existen sistemas con menor o mayor número de funcionalidades, que a pesar de tener ciertas estrategias implementadas para la integración, aún son consideradas insuficientes.

En Cuba, existen diversas Instituciones que cuentan con las tecnologías necesarias y el personal calificado para el desarrollo de la informática. Una de las instituciones de mayor avance tecnológico es la UCI, donde se produce una alta gama de software, tanto para clientes nacionales como internacionales y para distintas esferas como la educación, el deporte, entre otras. La UCI está llamada a convertirse en la principal potencia informática de desarrolladores de software del país, por lo que está enfocada en que el trabajo en los proyectos productivos sea satisfactorio, obtener productos con la calidad requerida y que

cumplan con las expectativas de los clientes que solicitan el software, pero para poder lograrlo necesita de una eficiente gestión de proyectos, lo cual hasta el momento no se ha cumplido completamente.

En el 2009 para realizar dicha función existían en la UCI varias herramientas como son el caso de: *Trac, REDMINE, Dot Project, Microsoft Project, Jira, Egroup Were.* Esta situación traía consigo una engorrosa gestión de proyectos y grandes problemas vinculados con la integración, puesto que no existía una sincronización entre las distintas herramientas dentro de la universidad. No estaban integrados los datos, por lo que la información se encontraba duplicada, desactualizada y no podía ser reutilizada. Las herramientas no se encontraban integradas, lo que evitaba el acceso e intercambio de información de los recursos existentes, los cuales son la base para el desarrollo de los proyectos. No existía una interoperabilidad entre los sistemas, por lo que no podían intercambiar información entre ellos, dando lugar a una limitación de las funcionalidades cruzadas de los mismos.

Dada esta necesidad y problemática para la gestión integrada de proyectos, se crea en la UCI el Sistema de Gestión de Proyectos (*GESPRO*). Es utilizado por más de seis mil usuarios y se encarga de la gestión de más de ciento cincuenta proyectos. Pero el sistema no cuenta con una buena descripción arquitectónica por lo que es necesario proponer una solución para la vista de integración.

A partir de lo anteriormente planteado surge como:

Problema a resolver:

Las insuficiencias existentes en la vista de integración de la Arquitectura del GESPRO 12.05 afecta la interoperabilidad del sistema.

Objeto de estudio:

La arquitectura del sistema GESPRO.

Objetivo general:

Proponer una solución para la vista de integración del sistema GESPRO 12.05 de la UCI.

Campo de acción:

La vista de integración de la arquitectura del sistema GESPRO 12.05.

Objetivos específicos:

1 Elaborar el marco teórico de la investigación.

2 Realizar propuesta de solución de la vista de arquitectura de integración del GESPRO 12.05.

3 Validar la vista de arquitectura de integración de la plataforma GESPRO 12.05.

Tipo de investigación:

Explicativa.

Tareas de investigación:

- ✓ Estudio del arte del tema que se investiga.
- ✓ Caracterización y selección de los estilos arquitectónicos.
- ✓ Caracterización y selección de los patrones y estrategias de diseño.
- ✓ Confección del estándar de codificación.
- ✓ Selección de las estrategias de integración e interoperabilidad.
- ✓ Validación de los resultados esperados.
- ✓ Montaje de la vista de integración de la arquitectura GESPRO 12.05 en la Suite GESPRO.

Hipótesis:

Si se obtiene una arquitectura para la vista de integración del sistema GESPRO 12.05 de la UCI, entonces se logrará la interoperabilidad entre este sistema y otros existentes.

Población:

Los centros de la red de producción de la UCI.

Muestra:

La red de centros de la sede central.

Variable dependiente:

La correcta interoperabilidad del sistema.

Variable independiente:

La arquitectura para la vista de integración del sistema GESPRO 12.05 de la UCI.

En el desarrollo del proceso investigativo, se emplearon varios métodos que se mencionan a continuación.

Métodos Teóricos:

Análisis y Síntesis: para estudiar el proceso de la información y arribar a las conclusiones de la investigación, así como precisar las características del modelo arquitectónico propuesto.

Histórico-Lógico: para determinar los antecedentes y las tendencias actuales del desarrollo de los modelos y enfoques arquitectónicos.

Métodos empíricos:

Observación: se usará debido a que es un procedimiento fácil de llevar a cabo y que permite percibir directamente, los hechos de la realidad objetiva para la percepción selectiva de las restricciones y propiedades del sistema y para el control de la evolución de la arquitectura inicial.

Aporte práctico

Como resultado se espera obtener la documentación y propuesta de la vista de integración del sistema GESPRO 12.05 y su correcta interoperabilidad, facilitando así la comunicación y el intercambio de información con otros sistemas.

Diseño de investigación

Pre experimento de Pre y Post prueba con un solo grupo, en este caso hay al menos un punto de referencia inicial. La validez interna puede ser afectada fácilmente por la historia, la maduración, la elección de un grupo atípico, la regresión y las interacciones.

Estructura capitular

El trabajo está estructurado en 3 capítulos, de los cuales se realiza una breve descripción a continuación:

Capítulo 1. **Fundamentación Teórica:** se establecen las bases teóricas del tema abordado en la investigación**.** Se muestran conceptos referentes a la Arquitectura de Software, como son, patrones y estilos arquitectónicos, así como su importancia.

Capítulo 2. Descripción de la arquitectura: en este capítulo se realiza la propuesta de solución al problema planteado en el diseño teórico de la investigación.

Capítulo 3. Evaluación de la arquitectura: está vinculado a la demostración de la implantación y validez de la arquitectura de la vista de integración para el sistema dado.

CAPÍTULO 1. Fundamentación teórica

1.1 Introducción

En el presente capítulo, se realiza el estudio del marco teórico referente al tema de la Arquitectura de Software. Se muestran los principales conceptos que se manejan en la misma, así como su importancia, y se realiza un análisis de los estilos arquitectónicos existentes y de los patrones de diseño.

1.3 Arquitectura de Software

En la actualidad, se puede encontrar gran cantidad de conceptos sobre arquitectura de software, puesto que no todos los escritores tienen la misma visión sobre el tema, basándose en diferentes conceptos que se muestran a continuación.

Una definición reconocida es la de Clements: "*la arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión del detalle inherente a la mayor parte de las abstracciones". (Reynoso, 2004).*

El documento de IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) 1471-2000, define que: 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 *(Reynoso, Carlos Billy, 2005).*

Al realizar un análisis de los conceptos abordados anteriormente, el autor determina que la Arquitectura de Software es la base para la creación de cualquier software, está orientada a un mejor diseño, a la evolución del sistema en desarrollo y a definir las relaciones de los componentes en el sistema. En la presente investigación, se plantea la

Arquitectura de Software según como lo define Clements, dicha definición detalla el concepto más completo.

1.5 Escuelas de la Arquitectura de Software

A continuación, una breve panorámica de las principales corrientes teóricas de la Arquitectura de Software.

- ✓ Arquitectura como etapa de ingeniería y diseño orientada a objetos.
- ✓ Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.
- ✓ Estructuralismo arquitectónico radical.
- ✓ Arquitectura basada en patrones.
- ✓ Arquitectura procesual.
- ✓ Arquitectura basada en escenarios.

1.5.1 Arquitectura como etapa de la ingeniería de software orientada a objetos

Esta corriente arquitectónica, se basa en el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman, combinando estrechamente el mundo de UML y Rational (*Abdurazik, Aynur, 2000*). En esta referencia la arquitectura se restringe a las fases preliminares del proceso, enfocándose en los niveles más elevados de abstracción. Importa más la abundancia, el detalle de diagramas y técnicas disponibles para la visión estructural del conjunto. La definición de arquitectura que se promueve en esta corriente, tiene que ver con aspectos formales a la hora del desarrollo. Las definiciones revelan, que la Arquitectura de Software en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descritas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten *(Kruchten., 1995).*

1.5.2 Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.

Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además, no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código *(Bass, Clements, 2003)*.

1.5.3 Estructuralismo arquitectónico radical

Se trata de un desprendimiento de la corriente anterior, mayoritariamente europeo, que asume una actitud más radical con el mundo UML *(Reynoso, Carlos Billy, 2005)*, *(Abdurazik, Aynur, 2000)*. En el seno de este movimiento hay al menos dos tendencias, una, que excluye de plano la relevancia del modelado orientado a objetos para la arquitectura de software y otra que procura definir nuevos meta-modelos y estereotipos de UML como correctivos de la situación. *(Jacopbson, Ivar, 1999).*

1.5.4 Arquitectura basada en patrones

Esta corriente se basa, principalmente, en la redefinición de los estilos como patrones arquitectónicos o de diseño *(Gamma, Richard Helm, 1995), (Buschmann Meunier, 1996)*. El diseño, consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura, con características de solución a problemas arquitectónicos comunes y clasificados.

1.5.5 Arquitectura procesual

Desde comienzos del siglo XXI, con centro en el Software Engineering Institute (SEI) y con participación de algunos de los arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci, Charles Weinstock. Intentan establecer modelos de ciclo de vida y técnicas de diseño de requerimientos, diseño, análisis, selección de alternativas, validación, comparación,

estimación de calidad y justificación económica específicas para la arquitectura de software. Otras variantes dentro de la corriente procesual se caracterizan por cambios en la etapa del proceso de extracción de la arquitectura, generalización y reutilización *(ERIKA CAMACHO, FABIO CARDESO, 2004).*

1.5.6 Arquitectura basada en escenarios

La Arquitectura basada en escenarios, es la corriente más nueva. Se trata de un movimiento predominantemente europeo con centro en Holanda. Recupera el nexo de la Arquitectura de Software con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. En esta corriente, suele utilizarse diagramas de casos de uso UML como herramienta ocasional, dado que los casos de uso son unos de los escenarios posibles y que no están orientados a objeto. Los autores vinculados con esta modalidad han sido, aparte de los codificadores de ATAM y demás métodos del SEI, los arquitectos holandeses de la Universidad Técnica de Eindhoven, de la Universidad Brije, de la Universidad de Groningen y de Philips Research: Mugurel Ionita, Dieter Hammer, Henk Obbink, Hans de Bruin, Hans Van Vliet, Eelke Folmer, Jilles Van Gurp y Jan Bosch. La presencia de holandeses es significativa; la Universidad de Eindhoven es incidentalmente, el lugar en el que surgió lo que P. I. Sharp proponía llamar la escuela arquitectónica de Dijkstra *(Reynoso, Carlos Billy, 2005).*

1.6 Vistas arquitectónicas

En los diferentes marcos estudiados, predomina como factor común de elementos conceptuales conductivos en el diseño y desarrollo de la arquitectura de software, las vistas arquitectónicas. Se conceptualiza Vista Arquitectónica como abstracción formalizada en prosa, lenguaje de modelado, gráfico o esquema informal, desde la que se describen las características arquitectónicas de un plano específico de la solución, según el interés de los involucrados *(Gamma, 1995).*

Las Vistas Arquitectónicas son la estructura central para la formalización y descripción de las arquitecturas, conforman la solución de software y están constituidas por los artefactos requeridos, las restricciones tecnológicas o de diseño a valorar.

La UCI, se dio a la tarea de confeccionar una nueva corriente arquitectónica para su beneficio, basándose en las anteriormente expuestas, tomando sus virtudes y tratando de mejorar las insuficiencias que presentan; apoyándose siempre en la utilización de las vistas arquitectónicas. Dicha corriente está compuesta por nueve vistas más una guía base que es utilizada para la construcción de diversos software.

Figura 1: Guía base de análisis arquitectónico

1.7 Estilos arquitectónicos

Dewayne Perry y Alexander Wolf, establecen el razonamiento de estilos de arquitectura como uno de los aspectos fundamentales de la arquitectura de software. Un estilo, es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de estilos cataloga las formas básicas viables de estructuras de software, mientras que las formas complejas se articulan mediante la composición de los estilos fundamentales *(Wolf, Dewayne, 2002).*

Cada estilo arquitectónico conjuga los componentes que realizan una función requerida por el sistema; los conectores que permiten la comunicación y cooperación entre dichos componentes; las restricciones que definen cómo estos pueden integrarse en el sistema; y modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes. Un software puede tener presente varios estilos arquitectónicos *(Wolf, Dewayne, 2002).*

De las definiciones estudiadas se concluye que, los estilos arquitectónicos constituyen la base fundamental de organización arquitectónica, a través de estos, se puede obtener el conocimiento necesario para identificar cómo estructurar los componentes y conectores que intervienen en el sistema.

1.7.1 Clases de estilos arquitectónicos

Los principales estilos arquitectónicos que se usan en la actualidad están divididos por Clases de Estilos que engloban una serie de estilos arquitectónicos específicos:

Estilos de Flujo de Datos

- ✓ Tubería y filtros

Estilos Centrados en Datos

- ✓ Arquitectura de Pizarra o Repositorio

Estilos de Llamada y Retorno

- ✓ Modelo Vista Controlador (MVC)
- ✓ Arquitecturas en Capas
- ✓ Arquitecturas Orientadas a Objetos
- ✓ Arquitecturas Basadas en Componentes

Estilos de Código Móvil

- ✓ Arquitectura de Máquinas Virtuales

Estilos Heterogéneos

- ✓ Sistemas Control de Procesos

- ✓ Arquitecturas Basadas en Atributos

Estilos Peer-to-Peer

- ✓ Arquitecturas Basadas en Eventos
- ✓ Arquitecturas Orientadas a Servicios
- ✓ Arquitecturas Basadas en Recursos

Estilos de Flujo de Datos

Esta clase de estilos enfatiza en la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros *(Billy y Kicillof, 2004).*

- ✓ Tubería y filtros

Ventajas*:*

- ✓ Es simple de entender e implementar.
- ✓ Fuerza un procesamiento secuencial.

Estilos Centrados en Datos

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra *(Billy y Kicillof, 2004).*

- ✓ Arquiteturas de Pizarra o Repositório

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. En base a esta distinción, se han definidos dos subcategorías principales del estilo:

1. Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar,

el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).

2. Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

Estilos de Llamada y Retorno

- ✓ Modelo Vista Controlador (MVC)
- ✓ Arquitecturas en Capas
- ✓ Arquitecturas Orientadas a Objetos
- ✓ Arquitecturas Basadas en Componentes

Modelo Vista Controlador

Se utiliza principalmente cuando es necesario modularizar la interfaz de usuario, las reglas de negocio y el control de eventos. Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: *(Billy y Kicillof, 2004).*

Modelo: administra el comportamiento y los datos del dominio de aplicación y responde a requerimientos de información sobre su estado. Mantiene el conocimiento del sistema, además no depende de ninguna vista y de ningún controlador.

Vista: maneja la visualización de la información.

Controlador: interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. Tiene tres variantes principales: Activa, Pasiva y Documento-Vista.

Ventajas:

- ✓ Soporte de vistas múltiples: dado que la vista se encuentra separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación Web pueden utilizar el mismo modelo

de objetos, mostrado de maneras diferentes.

- ✓ Adaptación al cambio: los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

Desventajas:

- ✓ Complejidad: el patrón introduce nuevos niveles de indirección y por lo tanto aumenta ligeramente la complejidad de la solución. También se profundiza la orientación a eventos del código de la interfaz de usuario, que puede llegar a ser difícil de depurar.

Estilo Arquitecturas en Capas

Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia. Se define el estilo en capas como una organización jerárquica, tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción. Los diagramas de sistemas clásicos en capas dibujaban las capas en adyacencia, sin conectores, flechas ni interfaces; en algunos casos se suele representar la naturaleza jerárquica del sistema en forma de círculos concéntricos *(Billy y Kicillof, 2004).*

Estilo Arquitecturas Orientadas a Objetos

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Si hubiera que resumir las características de las arquitecturas Orientadas a Objeto (OO), se podría decir que *(Billy y Kicillof, 2004)*:

- ✓ Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son así mismo las unidades de modelado, diseño e

implementación.

- ✓ Las interfaces están separadas de las implementaciones.

Estilo Arquitecturas Basadas en Componentes

Hay un buen número de definiciones de componentes, pero Clemens Alden Szyperski proporciona una que es bastante operativa: un componente de software, dice, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas. Que sea una unidad de composición y no de construcción quiere decir que no es preciso confeccionarla: se puede comprar hecha, o se puede producir en casa para que otras aplicaciones de la empresa la utilicen en sus propias composiciones. *(Billy y Kicillof, 2004).*

En un estilo de este tipo:

- ✓ Los componentes en el sentido estilístico son las unidades de modelado, diseño e implementación.
- ✓ Las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico.

Estilos de Código Móvil

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico. *(Billy y Kicillof, 2004).*

- ✓ Estilo Arquitectura de Máquinas Virtuales

La arquitectura de máquinas virtuales se ha llamado también intérpretes basados en tablas. De hecho, todo intérprete involucra una máquina virtual implementada en software. Se puede decir que un intérprete incluye un seudo-programa a interpretar y una máquina de interpretación. El seudo-programa a su vez incluye el programa mismo y el análogo que hace el intérprete de su estado de ejecución o registro de activación. La máquina de

interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución.

Estilos heterogéneos

Es la clase de estilos más fuertemente referida en los últimos tiempos, se incluyen en este grupo formas compuestas o indóciles a la clasificación en las categorías habituales. Podrían agregarse formas que aparecen esporádicamente en los estudios de nuevos estilos, como los sistemas de control de procesos industriales, sistemas de transición de estados, arquitecturas específicas de dominios o estilos derivados de otros estilos. *(Billy y Kicillof, 2004).*

- ✓ Sistemas de control de procesos
- ✓ Arquitecturas Basadas en Atributos

Sistemas de control de procesos

Desde el punto de vista arquitectónico, los sistemas de control de procesos se caracterizan no sólo por los tipos de componentes, sino por las relaciones que mantienen entre ellos. El objetivo de un sistema de esta clase es mantener ciertos valores dentro de ciertos rangos especificados, llamados puntos fijos o valores de calibración. Existen mecanismos tanto de retroalimentación *(feedback)* como de pre alimentación *(feedforward)*, y tanto reductores de oscilación como amplificadores; pero el tipo de retroalimentación negativa es el más común. La ventaja señalada para este estilo radica en su elasticidad ante perturbaciones externas. *(Billy y Kicillof, 2004).*

Arquitecturas Basadas en Atributos

La arquitectura basada en atributos o ABAS fue propuesta por Klein y Klazman. La intención de estos autores es asociar a la definición del estilo arquitectónico un framework de razonamiento ya sea cuantitativo o cualitativo basado en modelos de atributos específicos. Su objetivo se funda en la premisa que dicha asociación proporciona las bases para crear una disciplina de diseño arquitectónico, tornando el diseño en un proceso predecible. Con ello se lograría que la arquitectura de software estuviera más

cerca de ser una disciplina de ingeniería, aportando el beneficio esencial de la ingeniería al diseño arquitectónico. *(Billy y Kicillof, 2004)*.

Estilos Peer-to-Peer

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados. *(Billy y Kicillof, 2004).*

- ✓ Arquitecturas Basadas en Eventos
- ✓ Arquitecturas Orientadas a Servicios (SOA)
- ✓ Arquitecturas Basadas en Recursos (REST)

Estilos Peer-to-Peer

- ✓ Arquitecturas Basadas en Eventos

Características

- ✓ Flujo de datos en un sentido a través de redes de filtros.
- ✓ Requerimientos y respuestas entre clientes y servidores.

Ventajas del estilo

- ✓ Se optimiza el mantenimiento haciendo que procesos de negocios que no están relacionados sean independientes.
- ✓ Se puede agregar un componente registrándolo para los eventos del sistema; se pueden reemplazar componentes.

Desventajas del estilo:

- ✓ Un componente no puede utilizar los datos o el estado de otro componente para efectuar su tarea.

Estilos Peer-to-Peer

- ✓ Estilo Arquitecturas Orientadas a Servicios

Características

- ✓ Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas.
- ✓ Los componentes del estilo, o sea, los servicios están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran. Los servicios son las unidades de diseño e implementación.
- ✓ Los componentes que requieran un servicio pueden descubrirlo y utilizarlo dinámicamente mediante sus estándares sucesores. En general aunque hay alternativas no se mantiene persistencia de estado y tampoco se pretende que un servicio recuerde nada entre un requerimiento y el siguiente.

Estilos Peer-to-Peer

- ✓ Estilos Arquitecturas Basadas en Recursos

Define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web (WWW), donde los URIs identifican los recursos y HTTP es el protocolo de acceso. El argumento central de Fielding es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación. *(Billy y Kicillof, 2004).*

1.8 Estrategias de diseño arquitectónico

1.8.1 Basado en Artefactos

En esta estrategia la metodología de diseño se divide en tres fases, que son Análisis, Diseño del Sistema y Diseño de Objetos. En la fase de análisis se aplican tres técnicas de modelado que son modelado de objetos, modelado dinámico y modelado funcional. En la fase de diseño de sistema, tienen especial papel la llamada implementación de control de

software y la adopción de un marco de referencia arquitectónico, punto en el que se reconoce la existencia de varios prototipos que permiten ahorrar esfuerzos o se pueden tomar como puntos de partida. *(Bass, Clements, 2003)*.

1.8.2 Basado en Casos de Uso

El Proceso Unificado de Jacobson (PU), aplica una arquitectura orientada por casos de uso. La organización del proceso en el tiempo se define en fases. El PU se compone de seis fases: Modelado de Negocios, Requerimientos, Análisis, Diseño, Implementación y Prueba. Estos diagramas de flujo resultan en los siguientes modelos separados: modelo de negocio y dominio, modelo de caso de uso, modelo de análisis, modelo de diseño, modelo de implementación y modelo de prueba. El concepto de estilo puede caer en diversas coordenadas del modelo, en las cercanías de las fases y los modelos de análisis y diseño. *(Bass, Clements, 2003)*.

1.8.3 Basado en Línea de Producto

Comprende un conjunto de productos que comparten una colección de rasgos que satisfacen las necesidades de un determinado mercado o área de actividad. En la estrategia de arquitectura de Microsoft, este modelo está soportado por un profundo conjunto de lineamientos, herramientas y patrones arquitectónicos específicos, incluyendo patrones y modelos .NET para aplicaciones de línea de negocios, modelos aplicativos en capas como arquitecturas de referencia para industrias, etcétera. *(Bass, Clements, 2003)*.

1.8.4 Basado en Dominios

Considerado una extensión del anterior, se origina en una fase de análisis de dominio que puede ser definido como la identificación, la captura y la organización del conocimiento sobre el dominio del problema, con el objeto de hacerlo reutilizable en la creación de nuevos sistemas. El modelo del dominio se puede representar mediante distintas formas bien conocidas en ingeniería del conocimiento, tales como clases, diagramas de entidad-relación, redes semánticas y reglas. *(Bass, Clements, 2003)*.

1.8.5 Basado en Patrones

Esta estrategia busca codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. Los patrones de diseño se aplican en principio sólo en la fase de diseño, aunque a la larga la comunidad ha comenzado a definir y aplicar patrones en las otras etapas del proceso de desarrollo, desde la concepción arquitectónica inicial hasta la implementación del código. Se han definido, por ejemplo, lenguas o idiomas en la fase de implementación que mapean diseños orientados a objeto. *(Bass, Clements, 2003)*.

1.9 Patrones

El patrón es una descripción del problema y la esencia de su solución, de forma que la solución pueda reutilizarse en diferentes situaciones, no constituye una especificación detallada, es una solución adecuada a un problema común. Describe una solución probada a un problema recurrente de diseño, el cual ocurre en un contexto. Existen patrones de arquitectura, análisis y diseño. La definición de arquitectura requiere un proceso basado en el raciocinio sobre patrones. Un arquitecto reutiliza patrones para definir la estructura y el comportamiento interno y externo de un sistema. *(Reynoso, Carlos Billy, 2005).*

Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones. En el año 1977, Christopher Alexander expresa: cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puedes usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces" *(Sarver, 2000).*

A partir de los conceptos antes mencionado, se define como patrón, a la solución de problemas que ocurren repetidamente dentro de un contexto muy bien definido. Además se puede aplicar dichas soluciones en disímiles ocasiones sin tener que realizar una misma función varias veces.

1.9.1 Clasificación de los patrones

Los patrones pueden clasificarse o dividirse en:

Patrones de Arquitectura: relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento.
Según lo expresado en el texto de Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal, Pattern-Oriented Software Architecture (POSA), los patrones arquitectónicos son lo mismo que los estilos y ambos términos se usan de manera indistinta*. (Reynoso, Carlos Billy, 2005).*

Patrones de Diseño: fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC). En esta calificación encontramos los patrones GRAPS (General Responsibility Assignment Software Patterns) que describen los principios fundamentales de la asignación de responsabilidades a objetos y los GoF (Gang of Four) que enmarcan los llamados patrones de diseño Estructurales, Creacionales y de Comportamiento. *(Reynoso, Carlos Billy, 2005).*

Patrones de Análisis: usualmente específicos de aplicación, se aplican durante el análisis para solucionar problemas relacionados con el modelo de dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes. *(Reynoso, Carlos Billy, 2005).*

Patrones de Proceso o de Organización: tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización. *(Reynoso, Carlos Billy, 2005).*

Patrones de Idioma: se aplican durante las fases de desarrollo de implementación, despliegue y mantenimiento, constituyen estándares de codificación y proyecto, sumamente específicos de un lenguaje, plataforma o ambiente y están encaminados a

solucionar los problemas con la legibilidad, predictibilidad y operaciones comunes bien conocidas en un nuevo ambiente. Regulan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas *(Reynoso, Carlos Billy, 2005).*

Al contrario de los estilos arquitectónicos los patrones son muchos y a su vez muy variados, es casi imposible revisar todos los patrones que existen a la hora de hacer una determinada aplicación, por eso se recomienda el uso de los patrones que estén asociados en cada uno de los estilos que se seleccionen para el desarrollo de la arquitectura.

1.10 Patrones de diseño

1.10.1 Patrón Experto en Información:

El GRASP de experto en información es el principio básico de asignación de responsabilidades.

Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento) *(Reynoso, Carlos Billy, 2005)*.

1.10.2 Patrón Creador

El patrón creador ayuda a identificar quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases *(Reynoso, Carlos Billy, 2005).*

La nueva instancia deberá ser creada por la clase que:

1. Tiene la información necesaria para realizar la creación del objeto.
2. Usa directamente las instancias creadas del objeto.
3. Almacena o maneja varias instancias de la clase.

1.10.3 Patrón Controlador

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.

Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento. *(Reynoso, Carlos Billy, 2005).*

1.10.4 Patrón Alta Cohesión

Los conceptos de cohesión y acoplamiento están íntimamente relacionados. Un mayor grado de cohesión implica uno menor de acoplamiento. Maximizar el nivel de cohesión intramodular en todo el sistema resulta en una minimización del acoplamiento intermodular. *(Reynoso, Carlos Billy, 2005).*La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase.

1.10.5 Patrón Bajo Acoplamiento

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal manera que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. *(Reynoso, Carlos Billy, 2005).*

1.10.6 Patrón Polimorfismo

Siempre que se tenga que llevar a cabo una responsabilidad que dependa del tipo, se tiene que hacer uso del polimorfismo, cuando las alternativas o comportamientos relacionados varían según el tipo (clase), asigne la responsabilidad para el comportamiento utilizando operaciones polimórficas a los tipos para los que varía el comportamiento. Asigna el mismo nombre a servicios en diferentes objetos. *(Reynoso, Carlos Billy, 2005).*

1.10.9 Patrón Variaciones Protegidas

Es el principio fundamental de protegerse del cambio, de tal forma que todo lo que se prevé en un análisis previo que es susceptible de modificaciones, se envuelve en una

interfaz, utilizando el polimorfismo para crear varias implementaciones y posibilitar implementaciones futuras, de manera que quede lo menos ligado posible al sistema, de forma que cuando se produzca la variación, repercuta lo mínimo. *(Reynoso, Carlos Billy, 2005).*

1.11 Integración

Integrar es hacer que alguien o algo pase a formar parte de un todo. La integración recoge todos los elementos o aspectos de algo y lo incorpora al ente o a un conjunto de organismos. La Integración en la Arquitectura de Software busca una completa relación del espacio interior con el espacio exterior. Una dualidad que se complementa mutuamente con las características propias de cada ambiente o de cada plataforma operacional en el desarrollo de software. La Arquitectura de Integración persigue la obtención de una forma más eficiente y flexible de combinar recursos, con el objetivo de optimizar operaciones.

Implica integrar aplicaciones y fuentes de datos empresariales tal que puedan fácilmente compartir procesos de negocio e información. *(Sharma, 2003).*

La integración es la tarea de hacer trabajar juntas a aplicaciones dispares para producir un conjunto unificado de funcionalidad. *(Hohpe, 2003).*

Se expresa como el acto de combinar componentes de software (programas y archivos) en un sistema de software. En proyectos grandes se puede hablar de múltiples componentes pero que pueden llegar a ser sólo archivos de código de bajo nivel compilados para pequeños proyectos. (Duvall, 2007).

Aunque existen varios conceptos sobre el tema, lo cual implica que tengan algunas diferencias, muestran que la integración de sistemas involucra igualmente a información y procesos del negocio en función de proveer funcionalidad común.

Entre los niveles de integración se encuentran la interna y externa, la primera es la relación que existe entre los componentes o subsistemas dentro de una misma

organización, mientras que la integración externa es la relación que se establece con sistemas externos, ambas con el objetivo de juntar aplicaciones para lograr un conjunto de funcionalidades en común. A continuación la figura representada muestra un ejemplo general de ambas integraciones.

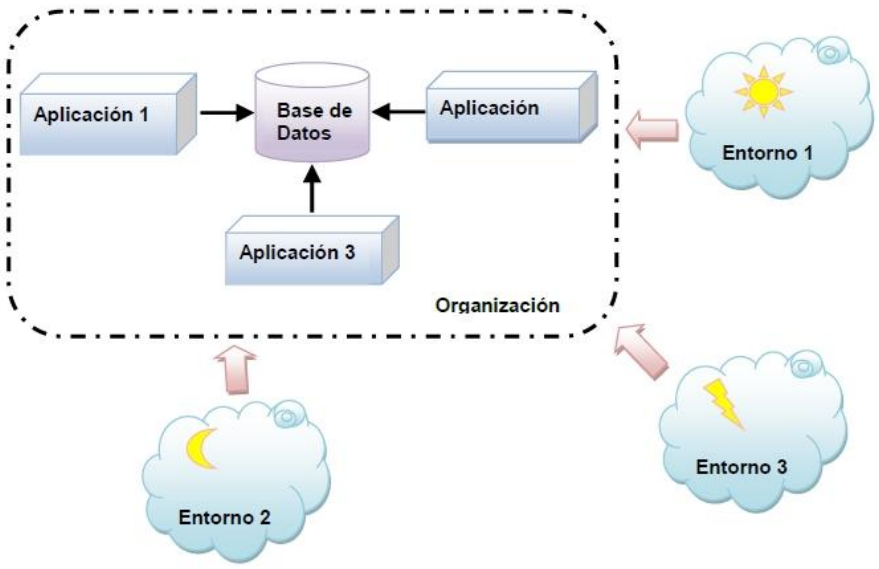

Figura 2 Integración interna y externa

1.12 Desafíos de la Integración de Sistemas

Se necesita establecer comunicación entre varios sistemas; además de que una vez integrados, cada unidad deja de controlar su única aplicación, puesto que esta pasa a ser parte del flujo general de información y servicios integrados. *(Linthicum, 1999).*

La integración tiene serias implicaciones en los sistemas involucrados. Una vez integrada una aplicación, cualquier proceso importante para una aplicación, se convierte en vital para todo el conjunto, por lo que una falla en este implicaría la falla de más de un sistema, lo cual pudiera tener consecuencias graves. *(Linthicum, 1999).*

El autor considera que al integrar varios sistemas, se debe tener mucho cuidado a la hora de manejar o realizar funcionalidades puesto que si alguna acción realizada

incorrectamente en alguno de ellos puede afectar grandemente el desarrollo y funcionamiento del otro sistema relacionado. Además las soluciones de integración necesitan ser escalables, dado que las relaciones con otras aplicaciones tienden a crecer en número con el paso del tiempo.

1.13 Interoperabilidad

Uno de los conceptos que va vinculado a la integración es la interoperabilidad, a continuación quedan plasmadas algunas de sus definiciones.

La IEEE la define como: "*La habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada*" *(IEEE, 1990)*.

El proyecto europeo IDABC (Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens) para la interoperabilidad y el establecimiento de servicios de gobierno electrónico a nivel europeo, define que la interoperabilidad es: "*La habilidad de organizaciones y sistemas dispares y diversos para trabajar juntos eficientemente para beneficiarse mutuamente en fines comunes*" *(IDABC EIF, 2008).*

La administración y el gobierno electrónico. La Interoperability Technical Framework (EITF) del Gobierno de Australia la caracteriza como "*La capacidad de transferir y utilizar información de una manera uniforme y eficiente a través de múltiples organizaciones y sistemas de tecnologías de la información*" *(IFWG, 2005).*

Después de un estudio de los conceptos antes expuestos, se dicide escoger para la investigacion el planteado por la IEEE, por ser esta una de las fuentes más confiables y que expone de manera sencilla el concepto más completo.

1.13.1 Niveles de Interoperabilidad

Desde la perspectiva de la defensa y la estrategia en los sistemas de información militar, Sandor Munk considera que la interoperabilidad tiene sentido en distintos dominios como: el físico, el de la información, el cognitivo y el social. Centrándose en la interoperabilidad

de la información, que define como la capacidad mutua de diferentes actores, necesaria para asegurar el intercambio y entendimiento común de la información que permita cooperar con éxito, Munk distingue tres niveles de capacidades:

1. **Nivel técnico**: conjunto de capacidades para la gestión de representaciones de materiales (físicas) que transportan información, lo cual es el fundamento y requisito esencial para el intercambio eficiente y exitoso de la información.
2. **Nivel sintáctico:** incluye capacidades para la gestión de representaciones intermediarias (no físicas) relacionadas con lenguajes, mensajes y formatos de datos empleados durante el intercambio de información.
3. **Nivel semántico**: grupo de capacidades relacionadas con el intercambio de diferentes representaciones de la información preservando su significado original.

1.13.2 Modelos de Interoperabilidad

Munk diferencia tres modelos de interoperabilidad teniendo en cuenta las características, el alcance de los sistemas que van a intercambiar información y el entorno que soporta la interoperabilidad:

1. **Modelo elemental**: se establece entre sistemas que pertenecen a la misma área funcional o especialización, y que están en una fuerte y permanente cooperación. Como consecuencia de la similitud de las funciones entre los sistemas participantes, y de la fuerte y permanente cooperación, generalmente estos sistemas manejan el mismo tipo de información y al intercambiar la misma puede ser definida fácilmente.
2. **Modelo complejo**: está también conectado con una colaboración relativamente permanente, pero cubre varias o todas las posibles áreas funcionales, y generalmente están soportados por esquemas o representaciones intermedias.
3. **Modelo global**: no está restringido a una cooperación dada, y describe la interoperabilidad entre estructuras y soluciones de intercambio de información en un entorno de información y cooperación cambiante.

1.13.3 Beneficios de la Interoperabilidad

La interoperabilidad entre los sistemas informáticos es de gran importancia, debido a su correcto funcionamiento se logra estandarizar la comunicación, se alcanza una mayor cooperación entre sistemas, se tiene la posibilidad de integrar sistemas en el futuro sin necesidad de grandes inversiones, se gana además, en aspectos como la sencillez en el intercambio de información influyendo positivamente en factores como el tiempo y el presupuesto. Permite la reutilización de datos, información y funcionalidades, mejorando así, los procesos de tomas de decisiones puesto que los datos son más confiables y poseen más calidad.

1.13.4 Estándares de interoperabilidad

Los estándares, "son acuerdos internacionales documentados o normas establecidas por consenso mundial. Contienen las especificaciones técnicas y de calidad que deben reunir todos los productos y servicios para cumplir satisfactoriamente con las necesidades para las que han sido creados y para poder competir internacionalmente en condiciones de igualdad" (Tortosa, 2006).

La interoperabilidad se basa en estándares, éstos establecen las reglas generales del juego para el equipo, dicen a dónde se quiere llegar, no cómo deben llegar. También sirven como guías aunque no son un manual de comportamiento.

En conclusión, un estándar es un modelo a seguir. Un estándar de interoperabilidad se especifica como tal en aspectos relacionados con la comunicación de procesos, y en la manera que se va a regir la organización y estructura de todo este trayecto a la comunicación y el intercambio entre sistemas.

A continuación se detallan algunos de los estándares de interoperabilidad.

SGML

Son las siglas en inglés para Standard Generalized Markup Language (lenguaje estándar de marcado generalizado). "*Sirve para especificar las reglas de etiquetado de documentos*

y no impone en sí ningún conjunto de etiquetas en especial" según *(Bastarrica, 2009).* SGML no es más que un estándar que define los métodos para la representación de la información que abarca otros estándares como XML y HTML, entre otros, siendo la raíz de esta gran familia.

XML (Extensible Markup Language) como lenguaje de intercambio de datos

"Es un lenguaje abierto que se ha desarrollado desde 1996 como un subconjunto de SGML y que ha sido adoptado por la W3C desde febrero de 1998. Permite describir el sentido o la semántica de los datos, pues separa el contenido de la presentación describiendo el contenido a través de etiquetas o marcas" *(Bastarrica, 2009).*

Un documento XML tiene varias formas de representación, el hecho de que un documento esté en este formato brinda muchas ventajas como portabilidad, depuración, independencia de la plataforma y sobre todo su edición para posteriores cambios en su estructura. Además, 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.

WSDL para la descripción de los contratos o interfaces de los servicios

"*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*" según *(Tortosa, 2006)*. Otros autores como Francisco Domínguez Mateos, *(Mateos, 2005)*, 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*". Se puede decir que WSDL no es más que un documento XML que describe a un servicio web o sus interfaces definiendo los mecanismos de acceso a los servicios web.

Protocolo REST (Representational State Transfer)

Según la bibliografía consultada REST, es un nuevo estilo arquitectónico 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. Tiene el potencial para permitir la construcción de aplicaciones distribuidas mucho más escalables e interoperables, compuestas por cientos de servicios desacoplados entre sí. Está fuertemente basado en HTTP 1.1; aunque es independiente de este protocolo, es el único utilizado masivamente diseñado para soportar los principios REST.

Protocolo SOAP (Simple Object Access Protocol)

Según *(Tortosa, 2006)* SOAP "*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 cómo organizar la información de una manera estructurada para intercambiarla entre los distintos sistemas".

Por otra parte *(Naranjo, 2007)* define SOAP como "*una arquitectura conceptual que organiza funciones de negocio como servicios interoperables permitiendo reutilización de servicios para satisfacer necesidades de negocio. Está basado en estándares y con independencia de los fabricantes*". Alejandro Botello *(Botello, 2005)* plantea que "*es un protocolo de mensajería para que puedan ser enviados a través de la red utilizando protocolos de transporte como HTTP, SMTP5"*. En otras palabras, es una arquitectura o estilo arquitectónico para la construcción de aplicaciones donde se juntan las tecnologías de la información con los procesos de negocio.

UDDI (Universal Description, Discovery and Integration)

Muchas son los autores que hacen referencia al estándar, por ejemplo, Salvador Otón Tortosa *(Tortosa, 2006)* plantea que UDDI "Es un directorio que contiene un registro o repositorio de descripciones de servicios web. Tiene dos objetivos fundamentales, primero servir de soporte a los desarrolladores para encontrar información sobre servicios web y

poder construir clientes; segundo, facilitar el enlace dinámico de servicios web, permitiendo consultar referencias y acceder a servicios de interés". Por otro lado, Pedro Espina Martínez, *(Martínez, 2007)*, plantea que UDDI está comúnmente considerado en la actualidad como la piedra angular de las Arquitecturas Orientadas a Servicios SOA, definiendo un método estándar de publicación y descubrimientos web.

Se puede resumir 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; se apoya en algunos estándares que han sido muy aceptados, tales como XML y SOAP entre otros.

SAML

SAML en su versión 2.0 es un estándar para intercambiar información de autenticación y autorización entre dominios. Está diseñado para ofrecer Single Sing-On (SSO) o (Sistemas de Autenticación Reducida) para interacciones automáticas o manuales entre sistemas. Éste permite el intercambio de información de autenticación y autorización sobre usuarios, dispositivos o cualquier entidad identificable llamados sujetos. Usando sintaxis de XML, SAML define el protocolo petición-respuesta por el cual los sistemas aceptan o rechazan sujetos basados en aserciones (world, 2010). SAML define la sintaxis y la semántica de las aserciones que se hacen acerca de un tema, por una entidad del sistema.

1.14 Conclusiones

En este capítulo quedaron expuestos los principales conceptos enmarcados en la Arquitectura de Software, se explica la importancia de realizar una buena arquitectura, se muestran las principales escuelas que estudian el tema, los estilos arquitectónicos, sus características, ventajas y desventajas, así como los patrones arquitectónicos y de diseño para asegurar una estructura que cumpla con todos los requerimientos del sistema. Se realiza además un estudio sobre los principales aspectos relacionados con la interoperabilidad entre sistemas así como de los estándares de interoperabilidad.

CAPÍTULO 2. Descripción de la arquitectura

2.1 Introducción

Para realizar la arquitectura para la vista de integración del sistema GESPRO 12.05, se necesita hacer una investigación sobre el mundo de la integración, así como de comprender por qué es necesaria, para posteriormente lograr un entendimiento de la arquitectura propuesta, se exponen en el presente capítulo los principales conceptos y definiciones asociados al tema y la propuesta de solución a la vista de integración de GESPRO.

2.2 Vista de arquitectura de integración para el sistema GESPRO 12.05

La creación de la vista de integración es de suma importancia para el sistema GESPRO; si se logra su correcta implantación se estará garantizando que el sistema sea interoperable, es decir, que pueda intercambiar información con otros sistemas que realizan funciones distintas y utilizar la misma. Para ello es preciso proponer las soluciones necesarias para la integración interna del sistema, así como para la externa, por lo cual se necesita la utilización de estándares y protocolos que garanticen la interoperabilidad.

Para la confección de la vista de arquitectura de integración para el sistema GESPRO 12.05 se deben tener en cuenta una serie de acciones que se mostrarán a continuación.

2.2.1 Selección de los estilos arquitectónicos a emplear

A continuación se seleccionan los estilos arquitectónicos que se encargan de estructurar y organizar arquitectónicamente el sistema, permiten la comunicación y cooperación entre los componentes de GESPRO.

Estilos de Flujo de Datos

- ✓ **Tubería y filtros**

Se ocupa de la reutilización y la modificabilidad de los datos del sistema. En GESPRO este estilo se utiliza en el proceso de cálculo, ya sea en la limpieza de datos como en la subida de información del GESPRO Gerencial al Transaccional. También se usa en las cadenas de eventos en las acciones que ejecutan las peticiones del framework Ruby on Rails. Es posible definir una cadena de eventos antes de la acción donde se pueden transformar los datos de los id de los proyectos copiados o incluso romper la cadena en cualquier punto de ella y de igual forma se puede definir una cadena de acciones luego de la acción y estas también pueden ser interrumpidas en cualquier momento y filtrada.

Estilos Centrados en Datos

- ✓ **Arquitectura de Pizarra o Repositorio**

El sistema cuenta con gran cantidad de datos perdurables y consistentes por lo que este estilo aporta la integrabilidad de los datos del sistema. Ejemplo de esto son los datos que guarda un determinado proyecto, gran parte de estos datos permanecerán en el sistema por un tiempo indefinido, posiblemente hasta que el proyecto se elimine por lo que con este estilo se garantiza su integrabilidad en el sistema GESPRO. El estilo se evidencia principalmente en la base de datos de proyectos terminados el cual guarda la información de los proyectos por un tiempo indefinido.

Estilos de Llamada y Retorno

- ✓ **Modelo Vista Controlador (MVC)**

Este estilo enfatiza la modificabilidad y la escalabilidad del sistema. Es el flujo de información que hay entre la interfaz del sistema y el almacenamiento. Soporta múltiples vistas lo que proporciona la ventaja de trabajar en varias ventanas y realizar distintas opciones. El sistema GESPRO se basa en el estilo MVC, separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes. Ejemplo, cuando se entra al sistema para registrarse se pone el usuario y contraseña en una vista, al aceptar, estos datos se envían a una controladora la cual envía al modelo donde se verifica que los datos entrados por el usuario sean correctos, de

serlos se envía al usuario hacia la vista correspondiente, si los datos son incorrectos se le muestra un mensaje. Este estilo también se vincula con la integración interna que existe entre el GESPRO Gerencial y el Transaccional, dado que las clases Projects, Issues e Indicators publican los datos de los proyectos, peticiones e indicadores en formato XML, donde el controlador publica los datos, el modelo es la forma de acceder a los datos y la vista muestra la información.

Estilos de Llamada y Retorno

- ✓ **Estilo Arquitecturas OO**

El lenguaje de programación que se utiliza es orientado a objeto. Los componentes del estilo se basan en principios OO, las interfaces del sistema están separadas de la implementación lo que garantiza que si hay cambios en el código no cambie considerablemente la interfaz y viceversa.

Estilos de Llamada y Retorno

- ✓ **Arquitecturas Basadas en Componentes**

Permite utilizar diversos componentes ya probados que no presentan errores para realizar distintas funciones. En el desarrollo del sistema se trata por todos los medios de no cambiar el código fuente del REDMINE como núcleo de GESPRO permitiendo poder integrarle al mismo cualquier plugin que se encuentre en el repositorio de componentes, el cual contiene los plugins del REDMINE, estos son código libre que están en la página oficial del REDMINE y GESPRO se nutre de ellos y los adapta a su manera y de acuerdo con sus necesidades con el objetivo de garantizar la calidad del sistema, un ejemplo de ellos es el plugin de importar tareas del Microsoft Project.

Estilos Peer-to-Peer

- ✓ **Estilo Arquitecturas Orientadas a Servicios**

Mediante el Plugin Sincronización automática, los proyectos, hitos de ejecución copiados de un GESPRO hacia otro, así como los indicadores que se calculan para realizar evaluaciones, seguimiento y control de los proyectos son actualizados automáticamente en un momento determinado establecido por el cliente de la aplicación mediante Servicios web, REST (formato XML), utilizando la funcionalidad REST API, que trae consigo las

últimas versiones de REDMINE, donde son compartidos algunos modelos de la aplicación, que pueden ser leídos y modificados según se estime conveniente. Esta interacción con el Web Service, será de una manera prescripta por su descripción utilizando mensajes SOAP, típicamente transportados usando HTTP con una serialización en XML que serán recibidos y procesados por la aplicación.

Estilos Peer-to-Peer

- ✓ **Arquitecturas Basadas en Recursos**

Este estilo define recursos identificables y métodos para acceder y manipular el estado de esos recursos. La suite GESPRO cuenta con el Plugin Gerencial, el cual copia los proyectos de un GESPRO para otro, así como los hitos de ejecución de los mismos, datos que serán actualizados periódicamente para poder darle control y seguimiento desde la alta gerencia a proyectos puntuales. Estas acciones se realizan haciendo uso del Servicio Web REST, que es un estilo característico de las arquitecturas basadas en recursos, donde se tienen objetos distribuidos en la red y se realizan interacciones con los mismos por medio de métodos para acceder y manipular el estado de estos recursos.

2.2.2 Selección de las estrategias de diseño

Una de las acciones de gran importancia para la confección de la vista arquitectónica es la selección de las estrategias de diseño a seguir, las cuales dan gran fortaleza y estructuración al diseño del sistema. A continuación se destacan las que se siguieron para el sistema GESPRO.

Estrategia de diseño arquitectónico basado en artefactos.

Esta estrategia se basa en la metodología de diseño que se lleva a cabo para el desarrollo del sistema, para la construcción del sistema GESPRO se utilizan las fases de desarrollo de software que son Análisis, Diseño del Sistema e implementación.

Estrategia de diseño arquitectónico basado en línea de producto.

Esta estrategia permite adaptar el sistema original a diferentes subsistemas. El sistema GESPRO está basado en esta estrategia de diseño puesto que cada uno de los centros

que dependen de la suite se basan en el original y pueden adaptar el mismo de acuerdo a sus necesidades ya sea agregando o quitando funcionalidades, esto también se aplica a otros escenarios donde se encuentra el sistema en explotación, como puede ser en DESOFT Empresa Nacional de Software y MININT.

Estrategia de diseño arquitectónico basado en dominios.

Es la captura y la organización del conocimiento sobre el dominio del problema, lo que quiere decir que el dominio o el área del conocimiento del sistema es la Gestión de Proyectos por lo cual solo trabaja en cuestiones referentes al tema.

Estrategia de diseño arquitectónico basado en patrones.

Esta estrategia se basa en la utilización de patrones de diseño de software con el fin de optimizar el código que se utiliza para la confección del sistema GESPRO y así diseñar la aplicación con un alto nivel de calidad, y en el menor tiempo posible.

2.2.3 Selección de patrones de diseño principales

Si se quiere contar con una buena arquitectura, debe de estar llena de patrones, estos dan solución a un problema dado y la misma puede reutilizarse en distintas situaciones. Los patrones se reutilizan para definir la estructura externa e interna del sistema. A continuación se seleccionan los principales patrones de diseño que se ponen de manifiesto en el sistema GESPRO.

Patrón Experto en información

Este patrón se ve reflejado en la suite GESPRO en la clase Project puesto que es la clase que conoce toda la información necesaria sobre los proyectos por lo que tiene la responsabilidad de la creación de un objeto o la implementación de un método de dicha clase. Gracias a este patrón se mantiene el encapsulamiento, los objetos utilizan su propia información para llevar a cabo sus tareas. Se distribuye el comportamiento entre las clases que contienen la información requerida. Son más fáciles de entender y mantener. La clase Project es una de las principales en la integración que se realiza entre los

sistemas de GESPRO, esto quiere decir que la clase Issue e Indicator adquieran menos importancia en temas en este tipo de integración.

En otro contexto, refiriéndose a la integración externa que la herramienta se integrará con los controladores de versiones GIT y SVN donde la clase repositorio es la encargada de manipular la información referente a cada uno de ellos.

Patrón Creador

La presencia de este patrón se ve reflejada en cada una de las clases que de una manera u otra intervienen en la interacción entre GESPRO y otra herramienta, ejemplo de esto se tiene: SVN y GIT, las cuales son clases que heredan de la clase Repository pero que se encargan de crear una instancia de SVN o GIT respectivamente, clases que intervienen en la integración entre GESPRO y los sistemas de control de versiones; además se tienen las clases Issue e Indicator y Project que son las encargados de crear una petición, un indicador o un proyecto respectivamente, clases que intervienen en la integración entre el GESPRO Transaccional y el Gerencial.

Este patrón se ve en la clase Issue del sistema ya que es la responsable de crear o instanciar nuevos objetos o clases. Esta clase es la que tiene la información necesaria para realizar la creación del objeto.

Patrón Controlador

El patrón controlador se ve reflejado en las clases ImportResourceController, ImportDepartmentController, ImportHumanResourcesController, ImportCostsController, ImportIssuesController, ImporterIntegratorController, ImportMsprojectController, estas clases son las encargadas de realizar la integración de todas las herramientas de Gestión de Proyectos que puedan exportar a formato .CSV, los datos relacionados con tareas, recursos, recursos humanos, departamentos y costos; así como en las clases ProjectController, IssueController, IndicatorController y RepositoryController, que están presentes en la integración de los diferentes tipos de GESPRO y con los diferentes sistemas de control de versiones respectivamente.

Patrón Alta cohesión

Este patrón se ve en la clase User, la información que almacena la clase debe de ser coherente y debe de estar relacionada con la clase. Este patrón permitirá tener clases fáciles de mantener, entender y reutilizar.

Patrón Bajo acoplamiento

Este patrón se ve reflejado en la clase Project, esta se encuentra lo menos ligada posible a otra clase. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

Patrón Polimorfismo

El patrón polimorfismo se ve reflejado en la siguiente clases: ApplicationController, dado que la misma posee varios métodos polimórficos los cuales son redefinidos en las clases hijas: ImportResourceController, ImportMsprojectController, ImportDepartmentController, ImportCostsController, ImportIssuesController, ImporterController, ProjectController, IssueController, IndicatorController y RepositoryController debido a que todas las clases controladoras heredan de esta debido a definiciones previas en el framework Ruby on Rails.

Patrón Variaciones protegidas

En el desarrollo de la aplicación GESPRO, se tiene la clase BaseImporterController encargada de encapsular varias de las tareas relacionadas con los métodos de importar desde las diferentes fuentes de datos y diferentes tipos de datos, para si en algún momento se define una nueva forma de importar datos, esta solo se agregue en dicha clase, para que las demás que heredan de ella como son (ImportResourceController, ImportHumanResourcesController, ImportDepartmentController, ImportCostsController, ImportIssuesController, ImporterIntegratorController, ImporterController), sufran el menor cambio posible y tengan a su disposición estas nuevas funcionalidades, lo mismo pasa si se elimina una funcionalidad creada que sea heredada a partir de esta por dichas clases.

2.2.4 Estándar de codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. El estándar de codificación es muy importante para los programadores por muchas razones, pues el producto final no es mantenido por ellos toda la vida del software, permite mejorar la lectura del software y que el equipo de trabajo encargado del mantenimiento y soporte del producto pueda entender el código con mayor facilidad. El código fuente debe resultar un entorno familiar para cada miembro del equipo, como si hubiese sido escrito por él mismo. El mejor método para asegurarse de que un equipo de programadores mantenga el código realizado con una gran calidad es estableciendo un estándar de codificación. Para el desarrollo de GESPRO se siguió el estándar de codificación definido para el lenguaje de programación de Ruby on Rails.

2.2.5 Integración Interna de GESPRO

La **integración interna** de GESPRO está marcada por la relación entre los subsistemas internos del mismo y se propone de la siguiente forma:

1. Integración entre la herramienta de gestión de proyectos REDMINE y el Paquete de Ayuda a la Toma de Decisiones y Soluciones Integrales *(PATDSI).* Se realiza mediante servicios web y a nivel de base de datos, donde el REDMINE mediante estos servicios puede acceder a los reportes que genera el Generador Dinámico de Reportes (GDR) del PATDSI, el cual consume los servicios del ChartServer y R-Server encargados de incluir gráficos en los reportes.
2. Integración de REDMINE con el cliente servidor de correo Zimbra. Esta integración se hace a nivel de ficheros, los cuales son un contenedor de información que son accedidos por el REDMINE, el cual se encarga de notificar al usuario en caso de que exista algún cambio, se haría mandándole correos al usuario con las modificaciones de las peticiones (asignar tarea, evaluación de la tarea, cambiar fecha de la tarea).
3. La Integración del GESPRO Transaccional con el GESPRO Gerencial. Es a nivel de servicios REST, logrando que se copien de un GESPRO hacia otro los proyectos, hitos de ejecución los cuales almacenan el tipo de petición y los

indicadores para el seguimiento, control y evaluación de los proyectos, estos indicadores están relacionados con las áreas de conocimiento de la dirección de proyectos: costo, tiempo, calidad. Para la evaluación de un proyecto existe el reporte Evaluación General de los Proyectos, el cual propone la evaluación cualitativa Bien (B), Regular (R) o Mal (M).

4. La Integración del GESPRO con el Sistema de Gestión Integral de Seguridad (ACAXIA), es mediante el protocolo SAML el cual ofrece privacidad, confidencialidad y anonimato a los involucrados, constituyendo este una facilidad para una buena administración y control de la información y medios materiales así como de garantizar la seguridad de forma centralizada del sistema GESPRO, dado en elementos como la autenticación y autorización que permiten conocer la identidad de la persona que entre al sistema.
5. Integración de GESPRO con el Gestor de Contenido Empresarial Alfresco. Se realiza mediante aplicación web embebida (AWB) ya que existe el plugin de gestión documental en GESPRO por el medio del cual se puede acceder a la información almacenada en el repositorio de Alfresco encargado de guardar toda la información relacionada con los expedientes de estudiantes y profesores.
6. La integración del GESPRO con el subsistema de ayuda es mediante un link, permitiendo un acceder a la página de ayuda la cual nos brinda un entendimiento básico sobre la estructura del sistema.
7. La integración del GESPRO con la base de datos de proyectos terminados es a nivel datos ya que por este medio accede y utiliza la información que se encuentra almacenada en dicha base de datos. Es fundamental a la hora de crear un nuevo proyecto, accediendo a la información de los proyectos que se encuentran en la base de datos con el objetivo de reutilizar la mayor cantidad de datos posibles, dígase el cronograma, la información o el proyecto en general.

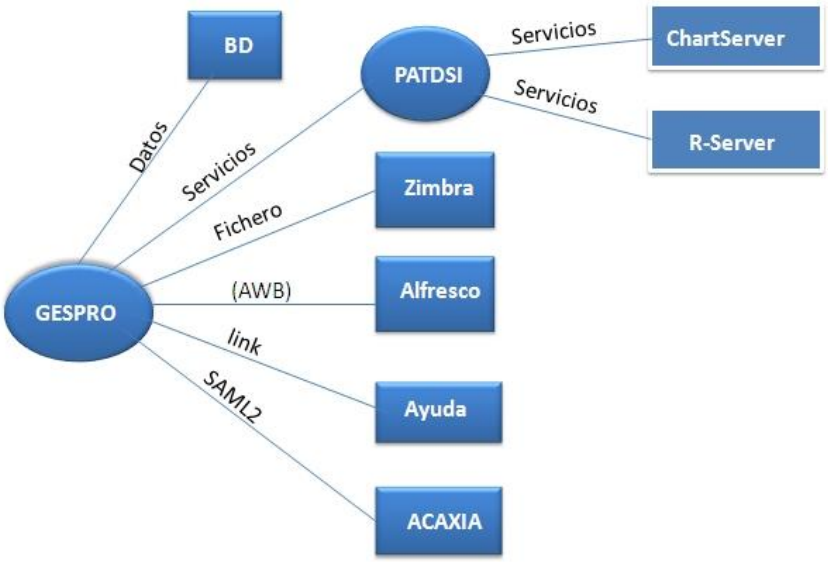

Figura 3 Nodo de integración interna

2.2.6 Integración Externa de GESPRO

Por otra parte la **integración externa** de GESPRO o también podríamos llamarla con el término de interoperabilidad está relacionada con los sistemas que se muestran a continuación y es posible gracias a que estos sistemas externos cumplen con los requerimientos mínimos para poder integrarse a GESPRO los cuales se evidencian en la utilización de estándares de interoperabilidad como WSDL, REST, XML para de esa forma poder establecer una comunicación entre ambos sistemas. A continuación se propone como quedará evidenciada la interoperabilidad del sistema.

1. Interoperabilidad con los sistemas de control de versiones (SVN y GIT). Esta integración se realiza mediante el protocolo SVN a través de HTTP combinado de aplicación web embebida a través de HTTP, permitiendo seguir y controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente, en la documentación y en las páginas web.
2. Interoperabilidad con el Sistema de Información Geográfica (SIG). El SIG se comunica con GESPRO a nivel de base de datos y por aplicación web embebida, accediendo a la base de datos de GESPRO y realizando con estos datos las distintas funciones para las que está concebido como mostrar la

información geográfica de la UCI y permite localizar geográficamente a centros y proyectos.

3. Interoperabilidad con el EXEL mediante ficheros .CSV, el cual es un fichero de texto donde la información se pone separada por coma para hacer fácil su entendimiento. En el EXEL se pueden gestionar proyectos los cuales son importados por el GESPRO.
4. Interoperabilidad a nivel de ficheros XML con el Microsoft Project (MS Project), herramienta ligera para gestionar proyectos, que facilita el uso de los diagramas de Gantt para la administración de cronogramas atendiendo a líneas bases, recursos, estimaciones de tareas y costos. El MS Project genera dichos ficheros y son cargados por el GESPRO.

Dado a la importancia que tiene la integración y las ventajas que brinda al sistema, se están trazando estrategias para lograr integrarse a otros sistemas como son el caso del Sistema Integral de Gestión CEDRUX y los sistemas de Planificación de Recursos Empresariales (ERP), ERP del MINFAR y open ERP. Dicha integración se realizará mediante ficheros XML, los cuales tienen la función principal de registrar las funcionalidades que ofrecen los métodos de las clases control de los componentes del sistema y a nivel de base de datos. De esta manera al integrarse con GESPRO, se producen un conjunto de facilidades como disminuir el margen de contaminación y repetición de la información y realizar el inventario.

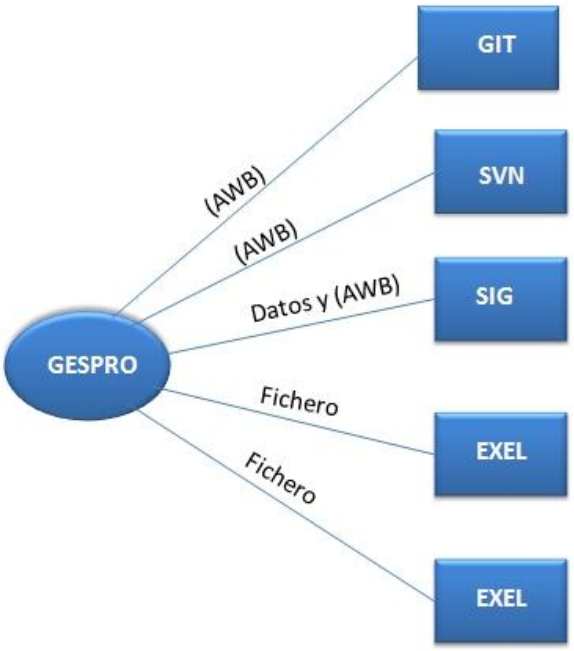

Figura 4 Nodo de integración externa.

2.7 Conclusiones

En este capítulo se presentó una propuesta de solución a la vista de arquitectura de integración del sistema GESPRO 12.05, para lograr dicha vista se dieron soluciones a una serie de acciones, que en su conjunto garantizan que exista una correcta integración entre los componentes internos del sistema y que GESPRO pueda relacionarse con otros sistemas ajenos a él.

Capítulo 3. Evaluación de la arquitectura

3.1 Introducción

Definir la arquitectura es uno de los primeros pasos en el ciclo de vida del desarrollo de un software. Estas decisiones son difíciles de cambiar una vez que el sistema se aplica, por lo que se deben efectuar las pruebas necesarias una vez definida la arquitectura. Los capítulos anteriores fundamentan desde el punto de vista teórico y describen una propuesta de arquitectura para la vista de integración del sistema GESPRO 12.05. Sin embargo, la misma pudiera considerarse incompleta, en ausencia de parámetros concretos que avalen su validez para resolver los problemas de la integración. El presente capítulo muestra los elementos y criterios que evalúan a la presente propuesta, los cuales a su vez sirven de base para demostrar su carácter válido, dando cumplimiento así a uno de los objetivos más importantes de esta investigación.

3.3 Aspectos para evaluar la arquitectura

Evaluar la Arquitectura de Software de un sistema permite detectar aquellos atributos de calidad que serán de vital importancia para el desarrollo del mismo, además de, obtener las metas específicas de calidad ante las cuales la arquitectura será juzgada; constituyendo la forma más económica de evitar todos los cambios que traería consigo el diseño de un sistema que no cumple los requerimientos necesarios. La evaluación de la arquitectura es un proceso iterativo e incremental que se puede realizar al final de cada ciclo. Existen dos variaciones útiles: temprana y tardía. *(YOAN ARLET CARRASCOSO PUEBLA, 2009).*

1. **La evaluación temprana**: es aquella que no tiene por qué esperar a que la arquitectura esté totalmente 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. *(YOAN ARLET CARRASCOSO PUEBLA, 2009).*

2. **La evaluación tardía**: toma lugar no solo cuando hay suficiente información de la arquitectura como para justificarlo, sino también cuando la implementación está completa. *(YOAN ARLET CARRASCOSO PUEBLA, 2009).*

En la evaluación de la arquitectura intervienen los mismos integrantes del equipo de desarrollo: el arquitecto, el diseñador y el administrador del proyecto, pero se puede contratar a un grupo de especialistas que realicen la evaluación deseada. El cliente también interviene de cierta forma en la evaluación, debido a que los resultados obtenidos son de su interés, puesto que decidirán si se continúa con la arquitectura prevista o si se hace necesaria la suspensión del proyecto.

Reglas de oro para determinar el momento de realizar evaluaciones:

- ✓ Realizar una evaluación cuando el equipo de desarrollo comienza a tomar decisiones que afectan directamente a la arquitectura. *(Salvador Gómez, 2007).*
- ✓ Cuando el costo de no tomar estas decisiones podría ser mayor que el costo de realizar una evaluación. *(Salvador Gómez, 2007).*

3.4 Atributos de calidad

Uno de los puntos críticos de cualquier evaluación de un software es sin dudas la elección de cuáles son los criterios básicos que regirán dicha revisión. Tanto es así que una selección errónea de estos criterios o atributos, conllevará necesariamente a consideraciones erróneas respecto al objeto evaluado, pudiendo esto tener serias consecuencias ulteriores en ambientes reales. La norma ISO/IEC 9126 en la parte primera de su revisión del 2001, establece un conjunto muy bien definido de atributos de calidad *(Clements, Paul, 2002)* que responden a características mundialmente aceptadas como deseables dentro de cualquier software y son introducidos a continuación.

Estos atributos de calidad pueden ser clasificados en dos categorías *(CAMACHO, CARDESO, 2004):*

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

Observables en tiempo de ejecución:

- ✓ **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.
- ✓ **Confidencialidad:** Es la medida de la habilidad de un sistema a 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.

No observables vía ejecución:

- ✓ **Configurabilidad:** posibilidad que se otorga a un usuario experto a 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:** capacidad de modificar el sistema de manera rápida y a bajo costo.
- ✓ **Portabilidad:** es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
- ✓ **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, de datos o procedimental.
- ✓ **Capacidad de Pruebas:** es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

3.5 Técnicas de evaluación de la Arquitectura de Software

Existen varias técnicas para evaluar las arquitecturas, las mismas se dividen en *cualitativas* y en *cuantitativas,* dentro de estos dos grupos se encuentran varias técnicas, que dependiendo del momento en que se realiza la evaluación y el tipo de resultado que se espera se debe escoger la más acorde a lo que se desea. Por lo regular, las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción; mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada. *(Clements, Kasman, 2002).*

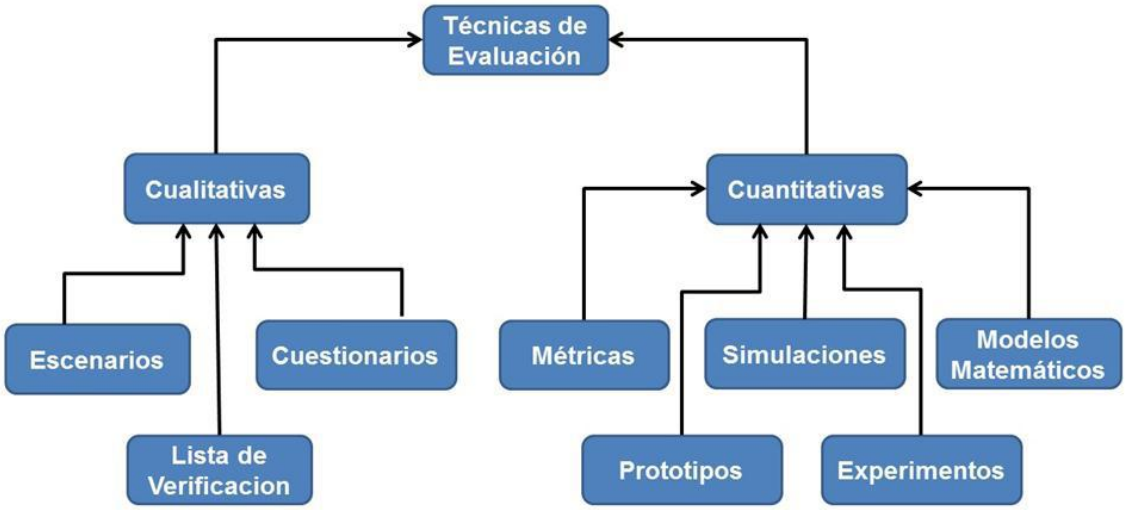

Figura 5: Clasificación de las Técnicas de Evaluación

3.5.1 Evaluación basada en escenarios

Los escenarios describen la interacción entre los *stakeholders* y el sistema, siendo los *stakeholders* cualquier involucrado en el desarrollo, desde clientes, desarrolladores, gerentes entre otros. Consta de tres partes: estimulo, contexto y respuesta. *(ERIKA CAMACHO, 2004).*

Estímulo: es la parte del escenario que explica o describe 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.

Contexto: describe qué sucede en el sistema al momento del estímulo.

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.

Los escenarios se definen en tres tipos:

- ✓ **Casos de uso:** involucran usos típicos del sistema y se utilizan para obtener información del mismo.

- ✓ **Escenarios de crecimiento:** tienen en cuenta posibles cambios futuros del sistema, por lo que cubren el principio de anticipación.
- ✓ **Escenarios de exploración:** cubren cambios extremos donde se espera que estresen el sistema.

Estos distintos casos de escenarios se emplean para probar el sistema de puntos de vistas diferentes, de manera que sean analizadas todas las decisiones arquitectónicas que puedan representar algún riesgo para el sistema. Algunas de las ventajas que presentan los escenarios es que son simples de crear y entender, efectivos, no son muy costosos y no necesitan de gran entrenamiento. Proveen un vehículo que permite concretar y entender atributos de calidad. Esta técnica basada en escenarios consta de dos instrumentos para la evaluación, el árbol de utilidades (u*tility tree*) que fue propuesto por Kazman y los perfiles (p*rofiles*) propuestos por Bosch.

3.5.2 Árbol de Utilidades

Para Kazman, los Árboles de Utilidades son esquemas que se representan en forma de un árbol en el cual presentan ciertos atributos de calidad de un sistema de software, donde se encuentran refinados hasta establecer escenarios con los cuales especifican detalladamente el nivel de prioridad que requiere o que tienen cada uno.

La intención del uso del Árbol de Utilidades es la identificación de los atributos de calidad más importantes para un proyecto particular. 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 Árbol de Utilidades contiene 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. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura *(ERIKA CAMACHO, 2004).*

3.5.3 Perfiles

Los perfiles constituyen conjuntos de escenarios en la mayoría de los casos presentan importancia relativa relacionada con cada escenario. El uso de ellos permite realizar especificaciones de manera más precisa del requerimiento para atributos de calidad. Estos constan de dos calcificaciones: (*ERIKA CAMACHO, 2004*).

- ✓ **Perfiles completos**: definen todos los escenarios relevantes como parte del perfil. Esto permite al ingeniero de software realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, puesto que incluye todos los posibles casos. Su uso se reduce a sistemas relativamente pequeños y sólo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad.

- ✓ **Perfiles seleccionados:** se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo a algunos requerimientos. La aleatoriedad no es totalmente cierta por limitaciones prácticas, por lo que se fuerza la realización de una selección estructurada de elementos para el conjunto de muestra.

3.5.4 Evaluación basada en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la Arquitectura de Software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad. *(Bosch, Jan, 2000).*

3.5.5 Evaluación basada en modelos matemáticos

La evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico y

se presenta como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados utilizando los resultados de uno como entrada para el otro *(Bosch, Jan, 2000)*. Entre las desventajas que presenta esta técnica se encuentra la inexistencia de modelos matemáticos apropiados para los atributos de calidad relevantes, y el hecho de que el desarrollo de un modelo de simulación completo puede requerir esfuerzos sustanciales.

3.5.6 Evaluación basada en experiencia

En muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en factores subjetivos como la intuición y la práctica, la mayoría logran ser justificadas por una línea de razonamiento y pueden ser la base de otros enfoques de evaluación *(Bosch, Jan, 2000)*. La evaluación basada en experiencia puede dividirse en dos grupos: la evaluación informal, que es la realizada por los arquitectos de software durante el proceso de diseño y la realizada por equipos externos de evaluación de arquitecturas.

3.6 Métodos de evaluación

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones. *(Clements, Paul, 2002)*. Seguidamente se abordan algunos de los métodos de evaluación.

3.6.1 Método de Análisis de Arquitectura de Software (SAAM)

SAAM es un método de evaluación basado en escenarios, que representa probables cambios a los que el sistema se encontrará sometido. Permite evaluar o comparar una o varias arquitecturas. Es usado para evaluar de forma rápida atributos de calidad como modificabilidad, escalabilidad, portabilidad e integridad.

Salidas de la evaluación del método SAAM:

- ✓ Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.
- ✓ Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación (*ERIKA CAMACHO, 2004*).

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones (*ERIKA CAMACHO, 2004*).

Pasos a seguir del método SAAM

- ✓ **Paso 1.** Desarrollo de escenarios.
- ✓ **Paso 2.** Descripción de la arquitectura.
- ✓ **Paso 3.** Clasificación de escenarios.
- ✓ **Paso 4.** Evaluación de escenarios.
- ✓ **Paso 5.** Interacción de escenarios.
- ✓ **Paso 6.** Evaluación general.

3.6.2 Método de Análisis de Acuerdos de Arquitectura de Software (ATAM)

Este método de evaluación indica cuán bien una arquitectura particular satisface las metas de calidad y provee ideas de cómo esas metas de calidad interactúan entre ellas (*ERIKA CAMACHO, 2004*).

El método de evaluación ATAM está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la

arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros (*ERIKA CAMACHO, 2004*).

Una de sus principales características consiste en identificar riesgos de la arquitectura diseñada a través de la identificación de puntos de sensibilidad, desventajas y riesgos. No cuenta con un modelo para el refinamiento de los atributos de calidad, pero si con el instrumento árbol de utilidad que presenta un nivel de refinamiento de los atributos más abarcadores. Este método es considerado como el más completo y es el encargado de analizar cómo la arquitectura logra satisfacer los atributos de calidad: modificabilidad, portabilidad, extensibilidad e integrabilidad y tiene en cuenta las dependencias que se crean entre ellos.

Este método consta de 9 pasos agrupados en 4 fases, las cuales se muestran a continuación:

Fase 1.Presentación

- ✓ **Paso 1.** Presentación del ATAM.
- ✓ **Paso 2.** Presentación de las metas del negocio.
- ✓ **Paso 3.** Presentación de la arquitectura.

Fase 2. Investigación y análisis

- ✓ **Paso4.** Identificación de los enfoques arquitectónicos.
- ✓ **Paso 5.** Generación del Utility Tree.
- ✓ **Paso 6.** Análisis de los enfoques arquitectónicos.

Fase 3. Pruebas

- ✓ **Paso 7.** Lluvia de ideas y establecimiento de prioridad de escenarios.
- ✓ **Paso 8.** Análisis de los enfoques arquitectónicos.

Fase 4. Reporte

- ✓ **Paso 9.** Presentación de los resultados.

3.6.3 Método de Revisión Intermedio de Diseño (ARID)

El método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura (*ERIKA CAMACHO, 2004*).

Es un método de bajo costo considerado por los autores como un híbrido entre Active Design Review (ADR), utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos y Architecture Trade-off Method (ATAM). En ADR, los involucrados reciben una documentación detallada y completan cuestionarios, cada uno de ellos por separado.

Kazman propone que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura (*ERIKA CAMACHO, 2004*).

Una de sus principales desventajas es en la manera de realizar la evaluación del diseño arquitectónico, donde se basa solamente en el análisis de los componentes de la arquitectura sin tener en cuenta las conexiones que se establecen entre ellos.

Este método consta de 9 pasos agrupados en dos fases que serán expuestas a continuación:

Fase 1. Actividades previas

- ✓ **Paso 1.** Identificación de los encargados de la revisión.
- ✓ **Paso 2.** Preparar el informe de diseño.
- ✓ **Paso 3.** Preparar los escenarios base.
- ✓ **Paso 4.** Preparar los materiales.

Fase 2. Revisión

- ✓ **Paso 5.** Presentación del ARID.

- ✓ **Paso 6.** Presentación del diseño.
- ✓ **Paso 7.** Lluvia de ideas y establecimiento de prioridad de escenarios.
- ✓ **Paso 8.** Aplicación de los escenarios.
- ✓ **Paso 9.** Resumen.

Pre experimento

Antes de evaluar la propuesta de arquitectura descrita, se decidió realizar un Pre experimento para comprobar la existencia de los atributos de calidad seleccionados con su historia anterior.

Atributo de calidad	Antes	Ahora
Integrabilidad	Existían varias herramientas de gestión de proyectos por separados, los cuales no representaban soluciones integrales para las organizaciones orientadas a proyectos o que manejan de alguna manera proyectos.	Se cuenta con un sistema integrado que posee una serie de plugins, los cuales pueden trabajar por separados, además de integrarse para compartir sus cualidades y realizar trabajo en conjunto. Puede integrarse con otros módulos o subsistemas, logrando así acceder a los datos e información de los mismos.
Escalabilidad	Los sistemas contaban con una pobre escalabilidad, puesto que eran herramientas concebidas para un objetivo y no permitían agregar	Se tiene un sistema basado en línea de productos que permite agregar nuevos componentes que cumplen con diversos objetivos, de acuerdo a las necesidades

	funcionalidades.	que presente el sistema, ampliando así el diseño arquitectónico inicial.
Interoperabilidad	No existía una interoperabilidad, puesto que cada sistema trabajaba por separado.	Se tiene un sistema que posee una alta interoperabilidad logrando comunicarse con otros sistemas y así intercambiar información y utilizar la información intercambiada. Algunos de los sistemas con los que se comunica son: Sistema de Información Geográfica (SIG), Sistema de Control de Versiones (SVN) y el Sistema de Gestión Documental (Alfresco).
Reusabilidad	No se podía reutilizar la información existente en los demás sistemas ya que no se tenía acceso a la misma.	Debido al nivel de integración existente, es posible reutilizar la información que se encuentra almacenada en los sistemas integrados al GESPRO puesto que se cuenta con una base de datos integra.

Tabla 1: Comparación de los atributos de calidad en el tiempo.

3.7 Evaluación de la propuesta de solución

Luego de realizar un estudio de los métodos de evaluación, se puede concluir que los mismos ayudan en la búsqueda de debilidades y soluciones del diseño arquitectónico seleccionado. Cada uno por separado es utilizado para evaluar la arquitectura de acuerdo a los elementos que se quiera evaluar de ella. Para evaluar la propuesta arquitectónica realizada en el capítulo dos se decidió utilizar de las técnicas cualitativas, la evaluación basada en escenarios. Se usó el instrumento Árbol de Utilidades, para identificar cuáles son los atributos de calidad relevantes y el método ATAM para evaluar la arquitectura y comprobar que satisface los atributos de calidad.

Para la aplicación del método ATAM sobre la arquitectura propuesta, fue realizada la presentación del mismo donde se abordó sobre su funcionamiento y principales características. Se expusieron los principales conceptos y el objetivo arquitectónico del método. Posteriormente se realizó la presentación de la arquitectura con el objetivo de verificar que la misma cumple con los atributos de calidad definidos: Interoperabilidad, Integrabilidad, escalabilidad y reusabilidad. Se analizaron los atributos de calidad mediante los cuales se obtuvieron los escenarios que a continuación son mostrados en el Árbol de Utilidades.

Árbol de Utilidades

El Árbol de Utilidades se encuentra formado por los atributos de calidad más importantes para la vista de arquitectura de integración del sistema GESPRO y sus escenarios.

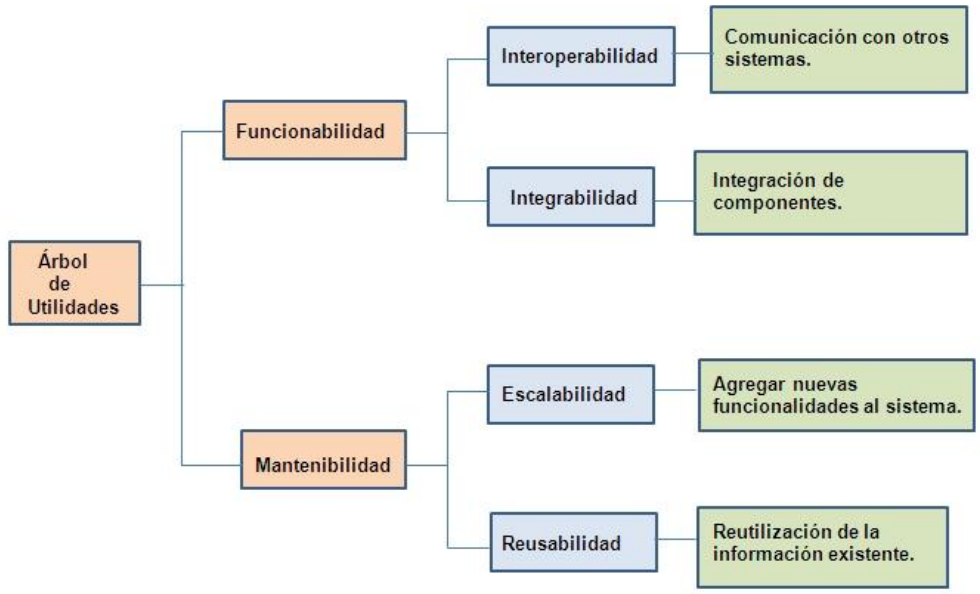

Figura 6: Árbol de Utilidades

Análisis de los escenarios

A continuación se muestran las diferentes tablas que resumen la relación entre los atributos de calidad y los escenarios definidos en el Árbol de Utilidades.

Escenario	Comunicación con otros sistemas.
Atributo	Interoperabilidad.
Ambiente	Se necesita interactuar con otros sistemas.
Estímulo	Se interactúa con otros sistemas.
Respuesta	Se logra la interacción con otro sistema, facilitando un intercambio de información.

Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Utilización de los estándares de interoperabilidad.				NR1
Uso del lenguaje de desarrollo Ruby.				NR2
Explicación	La interoperabilidad es la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada. El sistema GESPRO, tiene la capacidad de comunicarse con varios sistemas que fueron concebidos para realizar distintas funciones como son el caso del SIG y el SVN, la relación que se tiene con este último es la de leer los archivos que tiene el repositorio, lo cual se logra mediante los estándares de interoperabilidad y la implementación de funcionalidades a través del lenguaje Ruby para la integración de los sistemas.			

Tabla 2: Análisis del escenario "Comunicación con otros sistemas".

Escenario	Integración de componentes.
Atributo	Integrabilidad.
Ambiente	Se necesita la integración de nuevos componentes.
Estímulo	Se integran nuevos componentes al sistema.

Respuesta	Se logra la integración de nuevos componentes al sistema aumentando la funcionalidad el mismo.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso del estándar de codificación de Ruby para la implementación.				NR3
Explicación	Se integran al sistema una serie de componentes o plugins, los mismos pueden ser confeccionados por el equipo de desarrollo o adquiridos ya hechos y se encargan de realizar funciones de acuerdo al objetivo por el que fueron creados. La integración de los componentes no será un problema debido a que todos los programadores utilizarán un mismo estándar de codificación.			

Tabla 3: Análisis del escenario "Integración de componentes".

Escenario	Agregar nuevas funcionalidades al sistema.			
Atributo	Escalabilidad.			
Ambiente	Se requiere incorporar nuevas funcionalidades al sistema.			
Estímulo	Incorporar nuevas funcionalidades al sistema.			
Respuesta	Se agregan nuevas funcionalidades sin que se afecte el diseño del sistema.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**

Uso del lenguaje de desarrollo Ruby.				NR4
Explicación	Con la utilización del lenguaje de desarrollo Ruby se lograrán implementar nuevas funcionalidades que serán agregadas al sistema, lo cual no afecta el diseño del mismo y el estilo MVC que permite aumentar el diseño arquitectónico inicial. La decisión arquitectónica tomada constituye un no riesgo para el sistema.			

Tabla 4: Análisis del escenario "Agregar nuevas funcionalidades al sistema".

Escenario	Reutilización de la información existente.			
Atributo	Reusabilidad.			
Ambiente	Se requiere reutilizar los datos existentes.			
Estímulo	Se reutiliza la información.			
Respuesta	Se accede a la base de datos y se reutiliza la información.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso de PostgreSQL como gestos de base de datos.				NR5
Explicación	Con el almacenamiento de la información en el gestor de base de datos PostgreSQL se garantiza que la información pueda ser reutilizada por el GESPRO Gerencial y cada una de sus			

	modificaciones en los diferentes centros, la decisión arquitectónica tomada constituye un no riesgo pues no afecta al sistema.

Tabla 5: Análisis del escenario "Reutilización de la información existente".

La evaluación de la arquitectura demostró que se le da cumplimiento a los atributos de calidad Interoperabilidad, Integrabilidad, Escalabilidad y Reusabilidad enmarcados en la vista de arquitectura de integración del sistema GESPRO 12.05. Se encontraron cinco no riesgos, por lo que las decisiones arquitectónicas tomadas no ponen en peligro la propuesta arquitectónica.

3.8 Conclusiones

En este capítulo se realizó la evaluación de la arquitectura antes definida, fueron descritos los atributos de calidad, técnicas y métodos de evaluación para la arquitectura. Con la evaluación se comprobó que con la propuesta de solución dada se corrigen los problemas antes expuestos.

Conclusiones generales

En la realización de este trabajo:

- ✓ Se realizó un profundo estudio teórico, que favoreció en el entendimiento de los temas abordados sobre arquitectura de software.
- ✓ Se realizó la propuesta de la vista de arquitectura de integración del sistema GESPRO 12.05, logrando garantizar la interoperabilidad del sistema.
- ✓ Se validó la propuesta, determinando que la arquitectura definida da solución a los problemas planteados en la problemática, para lograrlo se describieron y seleccionaron los atributos de calidad, técnicas y métodos de evaluación eligiéndose la técnica escenarios con el instrumento Árbol de Utilidades y el método ATAM.

Recomendaciones

Luego de haber analizado los resultados del presente trabajo de diploma, resulta factible arribar a las siguientes recomendaciones:

- ✓ Continuar mejorando y actualizando el trabajo realizado para que sirva de referencia o guía de apoyo en el repositorio y lo pueda utilizar todo personal nuevo que se integre al proyecto.
- ✓ Considerar las posibilidades reales de extender la arquitectura propuesta a otras plataformas de desarrollo.
- ✓ Mejorar la integración con el sistema de control de versiones (SVN), para desde GESPRO poder subir archivos al mismo, así como poder crear una copia local del repositorio y realizar subida de archivos versionados para el SVN desde GESPRO.

Referencias bibliográficas

1- **Wolf, Dewayne, Perry, E. y L., Alexander. 2002.** *Foundations for the study of software architecture.* s.l.: ACM SIGSOFT Software Engineering Notes, 2002.

2- **Shaw, Mary y Garlan, David. 1996.** *Software Architecture: Perspectives on an Emerging Discipline.* s.l.: Pretince Hall, 1996.

3- **Jr, Frederick Brooks.** *The mythical man-month.* s.l.: Addison-Wesley, 1975.

4- **Reynoso, Carlos Billy** *Introducción a la Arquitectura de Software***.** 2005

5- **Jacopbson, Ivar.** *The Unified Software Development Process.* s.l.: Addison Wesley, 1999.

6- **Billy, Carlos y Kicillof, Nicolás** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft***.** 2004.

7- **Gamma, Richard Helm; Ralph Johnson; John Vlissides; Erich** (1995.), Design Patterns: Elements of reusable object-oriented software (Addison-Wesley).

8- **Sharma**, **Rahul, Stearns, Beth** y **Ng, Tony.** J2EE Connector Architecture and Enterprise Application Integration. s.l.: Addison – Wesley, 2003. 0201775808.

9- **Hohpe**, **Gregor** y **Woolf, Bobby**. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. s.l.: Addison - Wesley, 2003. 0321200683.

10- **Abdurazik, Aynur.** *Suitability of the UML as an Architecture Description Language with applications to testing.* s.l.: George Mason University, Febrero de 2000. Reporte ISE-TR-00-01.

11- **Kruchten, Philippe.** *The 4+1 View Model of Architecture.* s.l.: IEEE Software, Noviembre de 1995. pp. pp. 42-50.

12- **Bass, Len, Clements, Paul and Kazman, Rick.** *Software Architecture in Practice.* s.l.: Addison Wesley, 2003. 0-321-15495-9.

13- **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal.** *Pattern-oriented software. A system of patterns.* s.l.: John Wiley & Sons, 1996.

14- **ERIKA CAMACHO, FABIO CARDESO, GABRIEL NUÑEZ.** *ARQUITECTURAS DE SOFTWARE. GUÍA DE ESTUDIO.* Abril 2004.

15- **Multimedia, Glosario Curso de Multimedia. [En Línea]** http://moveframe.com/multimedia/index.phpoption=com_content&task=section&id=35&Itemid=216.

16- **Larsson, S. (2005).** IMPROVING SOFTWARE PRODUCT INTEGRATION. Department of Computer Science and Engineering Sweden, Malardalen University.

17- **Duvall, P. S. M., Andrew. (2007).** "Continuous Integration, Improving Software Quality and Reducing Risk -." from http://www.addisonwesley.de/main/main.asp?page=englisch/bookdetails&ProductID=121548

18- **Pressman, Roger S**. Ingeniería de Software: Un enfoque práctico. 2002.

19- **Sarver, Toby.** *Pattern Refactoring Workshop.* 2000. 27615.

20- **Gómez, Ing. Yoan Arlet Carrascoso Puebla e Ing. Enrique Chaviano.** Propuesta de arquitectura orientada a servicios para el módulo de inventario del ERP cubano. [En línea] 15 de 05 de 2009. http://www.gestiopolis.com/administracion-estrategia/erp-arquitectura-orientada-a-servicios.htm.

21- **Clements, Paul, Kasman, Rick y klein, Mark.** *Evaluating Software Architecture: Methods and Case Studies.* s.l.: Addison-Wesley, 2002.

22- **Bosch, Jan.** *Design and Use of Software Architectures.* s.l.: Addison Wesley, 2000.

23- **Salvador Gómez, Omar.** *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* s.l.: Brainworx S.A, 2007.

24- **Fowler, Martin.** *Patterns of Enterprise Application Architecture.* s.l.: Addison Wesley, 2003.

25- **Vazquez, Ivis Rosa.** *El Rol del Arquitecto de Software.* 2008.

26- **Institute of Electrical and Electronics Engineers.** *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York: s.n., 1990.

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

28- **Linthicum, David S.** *Enterprise Application Integration.* s.l.: Addison – Wesley, 1999.

29- **IDABC EIF**. 2008. Draft document as basis for EIF 2.0. [En línea] 2008.

30- **IFWG. 2005.** INTEROPERABILITY FRAMEWORK WORKING GROUP. *Australian Government Technical Interoperability Framework (AGTIF) [en línea]. Version 2. Commonwealth of.* [En línea] 2005. [Citado el: 16 de 01 de 2012.] http://www.finance.gov.au/publications/australian-government-technical-interoperability-framework/docs/AGTIF_V2_-_FINAL.pdf.

31- **Sandor Munk**, "Interoperability in the Infosphere. Challenges, Problems, Solutions". En: Jan Kamenícek. *Obrana a Strategie*, 2002.

32- **Tortosa, Salvador Otón. 2006.** *Propuesta de una arquitectura de software basada en servicios para la implementación de Repositorios Objetos de Aprendizaje Distribuidos.* 2006.

33- **Bastarrica, Maria Cecilia, Gutiérrez, Claudio y Ochoa, Sergio. 2009.** *Documentación electrónica e interoperabilidad de la información.* 2009.

34- **Ayala, Ramón Montero.** XML inicios y referencias.2005.

35- **Mateos, Francisco Domínguez. 2005.** *LIIBUS: Arquitectura de Sistemas Interoperables entre Middlewares Heterogéneos usando Máquinas Abstractas Reflectivas Orientadas a Objetos.* Departamento de Informática, Universidad de Oviedo. s.l.: Universidad de Oviedo, 2005.

36- **Botello, Alejandro. 2005.** *Arquitectura Orientada a Servicios (SOA).* s.l.: Universidad Politécnica de San Luis Potosí, 2005.

37- **Naranjo, Mauricio. 2007.** *Lineamientos para adopción de arquitectura orientada a servicios para las empresas.* Bogotá D.C.: LUCASIAN LABS, 2007.

38- **World, G.d. (2010).** Network World. Recuperado el Mayo de 2010, de http://www.networkworld.es/SAML-2.0-simplifica-la-federacion-de-identidades/seccion-/articulo-173366.

.