Universidad de las Ciencias Informáticas

Facultad 3

Título: Definición de un estilo arquitectónico para Xedro-ERP de la línea de productos de software Xedro.

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor (a): Zoemi Guerra Jardines.

Tutor (es): Ing. Yusnier Matos Arias.

Co-Tutor (es): M.Sc. Nemury Silega Martínez.
Ing. Andrés Reynaldo Milord.

Junio 2014

Declaración de Autoría

Declaro que soy el único autor del trabajo “Definición de un estilo arquitectónico para Xedro-ERP de la línea de productos de software Xedro” y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firma la presente a los ____ días del mes de _____ del año _____.

_____	_____
Zoemi Guerra Jardines	Ing. Yusnier Matos Arias
Autor	Tutor
_____	_____
M.Sc. Nemury Silega Martínez	Ing. Andrés Reynaldo Milord
Co-Tutor	Co-Tutor

Agradecimientos

Primeramente a mi mamá y a mi papá, por darme su apoyo incondicional y guiarme por el camino correcto. Sin sus consejos no hubiese logrado este sueño.

A mis segundos padres Ito y María que me han apoyado incondicionalmente en cada momento de estos cinco años.

A mi hermana por ser mi razón de ser.

A mi familia, especialmente a mi tía Teresa y a mi prima Iris Lidia por preocuparse por mí e impulsarme a seguir adelante.

A mi novio Samuel porque ha sido mi soporte, mi estímulo para seguir adelante, porque nunca me ha fallado, y sobre todo, por su paciencia para entenderme, a su familia que me han aceptado como soy y me hacen sentir como uno de ellos en especial a mi suegra Isis por ayudarme y preocuparse por mí en todo momento.

A mis tutores Nemury, Andrés y Yusnier, por tener mucha paciencia conmigo y ayudarme como lo han hecho.

A mis amistades y a todas las personas que he tenido el placer de conocer en esta maravillosa etapa de mi vida.

A todos mis compañeros de grupo, especialmente a Wilfredo, Reinier, Eddy y Eiler.

A todos mis profesores que jugaron un papel muy importante en mi formación como Ingeniera.

A todos los que de una forma u otra me ayudaron a transitar por mis estudios universitarios.

Dedicatoria

Quiero dedicar este trabajo especialmente a mis padres, por ser los principales impulsores de mi formación, por confiar en mí y sentirse tan orgullos de la hija que tienen.

Resumen

En el presente trabajo se expone una propuesta de estilo arquitectónico para la Línea de Productos de Software Xedro del Centro de Informatización de Gestión de Entidades, en la Universidad de las Ciencias Informáticas. Esta investigación se basa en la definición del vocabulario arquitectónico y las reglas que rigen las configuraciones de los elementos que conforman la arquitectura. Para lograr el objetivo propuesto de definir y describir el estilo arquitectónico para la línea Xedro, se partió de un estudio de los elementos relevantes que constituyen la misma.

Las Líneas de Productos de Software constituyen un modelo de desarrollo que tiene como particularidad fundamental que cada una de las aplicaciones que en ella se desarrollan comparte un conjunto común de características, las cuales satisfacen las necesidades específicas de un segmento particular de mercado. Las líneas de productos han sido aplicadas con éxito en muchas empresas a nivel mundial, y por consiguiente su uso oportuno podría traer resultados positivos a los proyectos que se realizan en la Universidad de las Ciencias Informáticas.

Palabras clave: estilo arquitectónico, línea de productos de software, vocabulario arquitectónico.

Tabla de Contenido

Índice de figuras 1
Índice de tablas 1
Introducción 1
Capítulo 1 Fundamentación teórica 6
1.1 Introducción 6
1.2 Líneas de productos de software 6
1.2.1 Los activos de una LPS y su arquitectura 8
1.3 Variabilidad en líneas de productos de software 9
1.3.1 Variabilidad en la arquitectura 10
1.4 Desarrollo basado en componentes 11
1.4.1 Sistemas basados en componentes 12
1.5 Arquitecturas y estilos arquitectónicos 13
1.5.1 Arquitectura de software 13
1.5.2 Estilos arquitectónicos 14
1.6 Conclusiones del capítulo 22
Capítulo 2: Estilo arquitectónico para la línea de productos 23
2.1 Introducción 23
2.2 Desarrollo del estilo arquitectónico 23
2.3 Definición del estilo 25
2.3.1 Vocabulario de elementos arquitectónicos 25
2.3.2 Reglas del estilo 26
2.4 Descripción del estilo 28
2.4.1 Descripción de las reglas del estilo 30
2.5 Conclusiones del capítulo 37
Capítulo 3: Validación de la solución 38
3.1 Introducción 38
3.2 Validación de la solución 38
3.2.1 Validación de las variables dependientes de la investigación 38
3.2.2 Validación de la variable independiente de la investigación 42
3.3 Resultados obtenidos 46
3.4 Premios otorgados a la solución 47
3.5 Conclusiones del capítulo 47

Conclusiones generales 48
Recomendaciones.....49
Bibliografía referenciada.....50
Bibliografía consultada 55
Glosario de términos 58
Anexos 60

Índice de figuras

Figura 1 Estilo arquitectónico Modelo-Vista-Controlador. 19
Figura 2 Un componente y un conector.. 24
Figura 3 Un conector.. 24
Figura 4 Modelo Conceptual del estilo arquitectónico. 36
Figura 5 Representación de la media aritmética de los indicadores a evaluar. 41
Figura 6 Representación de la media aritmética entre resultados individuales de los indicadores. 41
Figura 7 Diagrama de variabilidad y elementos comunes entre productos.. 44
Figura 8 Representación en porciento (%) de la evaluación de la métrica. 46
Figura 9 Conector evento en el caso de un administrador de eventos. 60
Figura 10 Conector evento en el caso de un bus de eventos.. 60

Índice de tablas

Tabla 1 Media de los indicadores evaluados. 42
Tabla 2 Evaluación de los indicadores definidos por el especialista # 1. 61
Tabla 3 Evaluación de los indicadores definidos por el especialista # 2. 61
Tabla 4 Evaluación de los indicadores definidos por el especialista # 3. 61
Tabla 5 Evaluación de los indicadores definidos por el especialista # 4. 62
Tabla 6 Evaluación de los indicadores definidos por el especialista # 5. 62
Tabla 7 Evaluación de los indicadores definidos por el especialista # 6. 62
Tabla 8 Evaluación de los indicadores definidos por el especialista # 7. 63

Introducción

En la actualidad las empresas se enfrentan a la competencia, el avance tecnológico y el aumento de la información a procesar en aras de obtener mejores resultados (KIM, y otros, 2005). Con el avance de las tecnologías en el campo de la informática, los procesos de desarrollo de software se hacen cada vez más complicados. Un componente dentro de este proceso que constituye una solución a muchos de los problemas es la arquitectura de software, que brinda una visión global del sistema y la relación entre las partes del mismo. Aparejado con la arquitectura también han ido avanzando en desarrollo y cualidades los estilos y patrones, los cuales de cierta forma responden a problemas que se ven a diario en la sociedad actual. Hoy en día existen diversas variantes en cuanto a estilos de arquitecturas, cada uno de los cuales defiende un criterio específico.

En los últimos años se ha podido observar un incremento en la demanda de *software* y en su complejidad requerida. Es por eso que hoy, para poder alcanzar los niveles deseados de calidad y mejorar la productividad, muchos sistemas se están desarrollando basándose en la ingeniería de Línea de Productos de Software (LPS), las cuales ayudan a reducir significativamente el tiempo de puesta en producción y los costos de desarrollo, mediante la reutilización de todo tipo de artefactos (Bosch, 2000; Clements, y otros, 2002). Esto hace que el desarrollo basado en LPS sea complejo, pero permite un desarrollo intensivo y extensivo de *software* gracias a la reutilización y variabilidad de los artefactos que se utilizan.

En (Clements, 1996) se plantea que una LPS consiste en un conjunto de sistemas software, que comparten un conjunto común de características (*features*), las cuales satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base (*core assets*) de una manera preestablecida.

En la Universidad de las Ciencias Informáticas (UCI), en la facultad 3 se encuentra el Centro de Informatización de Gestión de Entidades (CEIGE). En este centro se está optando por un enfoque basado en LPS, donde la línea de productos Xedro se encarga del desarrollo de sistemas integrales de gestión para entidades presupuestadas y empresariales que se rigen por las normas establecidas en Cuba. La misma tiene como basamento teórico al sistema

desarrollado anteriormente en el propio centro, llamado Xedro-ERP, que es un sistema integrado de gestión que se encarga de la planificación de recursos empresariales.

Como parte de una estrategia propia del centro, se llevó a cabo un análisis a la línea de productos Xedro, para ello se designó algunos integrantes que llevarían a cabo esta revisión. El resultado de dicho análisis arrojó que existen problemas que afectan al buen desenvolvimiento del equipo de desarrollo, estos problemas están dados a que:

- Los productos son desarrollados por separados sin tener en cuenta los puntos en común.
- Los productos se desarrollan sin seguir una arquitectura común, es decir cada producto tiene una arquitectura propia y no hay una uniformidad en el análisis de los mismos.

Existe poca reutilización de los artefactos utilizados en los productos de la línea. Básicamente cada producto se diseña aislado del resto y no se ha logrado identificar conceptos unificadores en los productos. Por ejemplo, en el Expediente de proyecto de los productos de Xedro-ERP, documento “Arquitectura de Software”, se registra que en los subsistemas Banco, Caja, COPA no existe reutilización de componentes. También en Contabilidad se registra que no se reutilizaron otros componentes o elementos externos en la solución del subsistema.

Otro efecto negativo se produce al crear diferentes instancias de arquitecturas y crear la estructura del sistema, los productos a desarrollar no toman como referencia el mismo estilo arquitectónico por lo que no hay una uniformidad en el análisis de los mismos.

Entre las personalizaciones que se han realizado en el centro está la personalización Sistema de gestión de inventarios para la gran misión viviendas, de Venezuela, Sistema integral de gestión para el Grupo Empresarial de la Industria Ligera (GEIL), entre otras.

Como resultado de las encuestas realizadas a algunos de los implicados en esta personalización, fundamentalmente desempeñando roles como: arquitecto, desarrollador, jefe de proyecto, se han identificado problemas asociados a malas prácticas que se han seguido para la integración de los componentes. Entre las causas de estas se pueden mencionar:

- Indefiniciones de la forma correcta para integrar componentes.

- Indefiniciones de los tipos de elementos que intervienen en la arquitectura así como sus interacciones.

También recientemente, en marzo de este mismo año, se inició la personalización a GEIL. En el cual se agregó una funcionalidad al módulo Contabilidad que se basa en asignar un tipo de cuenta, especificando como se va a cerrar la cuenta nominal (por cierre de ejercicio o por cierre de período). Además en Estructura y Composición se agregó, que en el nomenclador tipo de cifra los campos a llenar fueran opcionales. Sin embargo, fue cancelada esta personalización por parte del cliente en abril, no siendo útil para la presente investigación.

A partir de todos los elementos planteados anteriormente, se identificó como **problema a resolver:** ¿Cómo aumentar la reutilización e integrabilidad de la Línea de Productos de Software Xedro?

Con el objetivo de solucionar el problema planteado anteriormente el **objeto de estudio** queda enmarcado en el proceso de definición de estilos arquitectónicos, y delimitando el **campo de acción** en el proceso de definición de un estilo arquitectónico para la Línea de Productos de Software Xedro.

Para llevar a cabo este trabajo se planteó como **objetivo general:** Definir un estilo arquitectónico para la línea de productos Xedro, que permita aumentar su grado de variabilidad y así contribuya a aumentar el nivel de reutilización e integrabilidad de los productos.

Para dar cumplimiento al objetivo general se definieron los siguientes **objetivos específicos:**

- Establecer el marco teórico de la investigación en lo relativo a estilos arquitectónicos así como patrones y tácticas de variabilidad en líneas de productos de software, para sentar las bases teóricas del trabajo.
- Definir el estilo a partir de patrones y tácticas de variabilidad en líneas de productos de software.
- Validar la propuesta de solución.

Proponiéndose como **idea a defender**: con la definición y descripción del estilo arquitectónico se aumenta el nivel de reutilización e integrabilidad de los productos de la línea de productos Xedro.

Aporte:

- Catálogo de patrones de variabilidad para líneas de productos aplicados a Xedro.
- Estilo arquitectónico para la línea de productos Xedro.

Durante el desarrollo de esta investigación se han empleado varios métodos. Los métodos teóricos empleados son:

- **Histórico – Lógico:** para determinar las tendencias en la bibliografía; en particular, se ha utilizado durante el estudio del estado del arte para realizar un seguimiento evolutivo de los conceptos de arquitectura de software. Además para determinar las tendencias actuales de los estilos arquitectónicos.
- **Analítico – Sintético:** para llegar a conclusiones en la investigación a partir de la información que se procese y precisar características del estilo arquitectónico propuesto.

De los métodos empíricos se utilizaron:

- **Encuestas:** se utilizaron para obtener criterios acerca del estilo arquitectónico propuesto y para evaluar los resultados.
- **Entrevistas:** este método se utilizó para validar la propuesta, al entrevistar al personal especializado en temáticas como la arquitectura de software.

La estructura del trabajo de diploma será la siguiente:

Capítulo 1: Fundamentación teórica: En este capítulo se hace un estudio en relación a las LPS y la arquitectura de software. Además se realiza un estudio del desarrollo basado en componentes debido a que se utiliza el mismo principio básico que en las LPS. Por último, se presenta una revisión de varios estilos arquitectónicos que son relevantes para el objetivo de esta tesis.

Capítulo 2: Estilo arquitectónico para la línea de productos: En este capítulo se realiza la propuesta de solución al problema planteado en el diseño teórico de la investigación. Se presenta el estilo arquitectónico de la línea de productos Xedro, a partir del proceso de

construcción de un estilo arquitectónico definido por el Dr.Sc. David Garlan en su libro “Formal Modeling and Analysis of Software Arquitecture” del 2003.

Capítulo 3: Validación de la solución: En este capítulo se muestran los resultados obtenidos de la validación de la variable dependiente e independiente, así como una valoración sobre el mismo.

Capítulo 1 Fundamentación teórica

1.1 Introducción

En este capítulo se abordan los fundamentos teóricos asociados a la investigación, se hace un estudio de la literatura sobre LPS y arquitecturas de software. Se comienza con una revisión de las LPS, en la cual se brindan los conceptos más relevantes para este trabajo, incluyendo la variabilidad de los productos. Luego se hace un estudio de la arquitectura de software, en la cual se exponen las definiciones de arquitectura, y estilo arquitectónico. Por último, se presenta una revisión de varios estilos arquitectónicos que son relevantes para el objetivo de esta tesis.

1.2 Líneas de productos de software

Una LPS se basa en la creación de varios productos similares a partir del ensamblaje de partes de software previamente elaboradas (Montilva, 2006). Está fundamentada en la reutilización de software como aspecto clave para reducir los costos, aumentar la productividad y asegurar la calidad de los productos.

En (Montilva, 2006), la reutilización de software en las líneas de productos se caracterizan de la siguiente forma:

- Estratégica, donde se consolida lo común entre la línea de productos, maneja estratégicamente la variación entre los productos de la línea y elimina la duplicación de esfuerzos de ingeniería.
- Predictiva, en el cual la reutilización de activos se da en uno o más productos sobre una línea bien definida y se reutilizan arquitecturas de software, en lugar de reutilizar componentes de manera oportunista.
- Gestionada, donde es sistemática, planificada, institucionalizada y mejorada.

El creciente desarrollo de las LPS no se produce únicamente por la reutilización de software, sino por los beneficios notables que se le atribuyen. Entre los cuales se encuentran los que se detallan en (Montilva, 2006) y (Casallas):

- El aumento de la productividad.
- Reducción del tiempo para salir al mercado.
- Aumento de la calidad del producto.

- Mayor satisfacción del cliente.
- Capacidad para llevar a cabo la personalización en masa del producto.
- Reducción en el costo de producción de los productos.
- La capacidad para mantener la presencia en el mercado.

Las líneas de productos de software según Clements y Northrop en (Clements, y otros, 2001) se definen como: "Una línea de productos de software es un conjunto de sistemas software, que comparten un conjunto común de características (*features*), las cuales satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base (*core assets*) de una manera preestablecida".

Esta definición agrupa cinco conceptos fundamentales de las LPS (Díaz, y otros, 2010), a saber:

- "…un conjunto de sistemas software…": el objetivo de una LPS no es el desarrollo de un producto, sino el de un conjunto de productos. Hay que delimitar y preestablecer en la medida de lo posible el alcance de este conjunto.
- "…que comparten un conjunto común de características (*features*)…": este conjunto de productos se define por medio de características. Una característica es una peculiaridad del producto que los clientes consideran importante para describir y distinguir entre los distintos miembros de la LPS. Por ejemplo, en los ordenadores, estamos acostumbrados a caracterizarlos en términos del procesador, las dimensiones de la pantalla, etc. Todas ellas son características que sirven para distinguir los productos, y que son significativas para el usuario final. Se han documentado casos de LPS con más de 10 000 características. En estos casos es fundamental disponer de algún modelo que nos ayude a organizar las características, y especificar sus dependencias. Son los modelos de características.
- "…satisfacen las necesidades específicas de un dominio o segmento particular de mercado…": una LPS se desarrolla orientándose a un segmento de mercado concreto, es decir, los productos intentan satisfacer las necesidades específicas de un segmento de mercado. De la habilidad para acotar e identificar correctamente este mercado, dependerá el éxito de la LPS.

- "…se desarrollan a partir de un sistema común de activos base (*core assets*)…": los productos son desarrollados a partir de un conjunto común de activos reutilizables. Este término engloba la diversidad de elementos, tales como requisitos, planificaciones, modelo de características, arquitecturas, componentes, código fuente, etc., que conforman la base sobre la que se construye el producto. El reto está en determinar no sólo lo común sino también lo que se va a permitir variar.
- "…de una manera preestablecida.": los productos se construyen de una forma preestablecida, es decir, no sólo hay elementos comunes, sino que la estrategia para construir el producto está perfectamente establecida de antemano a través de un plan de producción.

1.2.1 Los activos de una LPS y su arquitectura

Los activos de una LPS son todos los artefactos producidos durante el desarrollo del software y que son potencialmente reutilizables, algunos ejemplos de activos son: patrones de diseño, componentes de software, manuales de usuario, especificación de requisitos, entre otros.

En (Clements, y otros, 2001; Oliveira, y otros, 2005) se plantea que el principal activo de una LPS es la arquitectura. La arquitectura define no sólo los atributos de calidad sino que también comprende la capacidad de reutilización, la derivación del producto, y la evolución de la línea de productos.

La arquitectura describe la estructura de toda la línea de productos y no solamente la de un producto particular, captura los aspectos comunes y variables, en el cual los aspectos comunes de la arquitectura son capturados por los componentes de software que son comunes a toda la línea y los aspectos variables de la arquitectura son capturados por los componentes de software que varían entre los miembros de la línea.

La arquitectura de una línea de productos es distinta a una arquitectura 'típica' pues para permitir la construcción de distintos productos por encima de ella, debe definirse una serie de puntos de variación que son necesarios para poder crear los distintos productos. La arquitectura también debe asegurar que existan mecanismos de variabilidad en los productos, donde para una LPS las formas de presentación de la variabilidad dentro de la arquitectura es esencial (Krueger, 2007).

Teniendo en cuenta los conceptos estudiados anteriormente, esta investigación apoya la idea de Clements y Oliveira referente a que la arquitectura de las LPS es el activo más importante de la línea, ya que la arquitectura de software es la organización estructural que garantiza por el cual se va a sustentar toda la solución. El estudio de las LPS aporta beneficios significativos a la solución pues es el intento más reciente de reutilización de software, donde intenta construir sistemas mediante el ensamblaje de componentes previamente fabricados.

1.3 Variabilidad en líneas de productos de software

La variabilidad tiene gran significación en el desarrollo de LPS, ya que posibilita el reúso constructivo y facilita la derivación de diferentes productos, específicos para cada cliente de la línea de producto (Pohl, y otros, 2003).

La administración de variabilidad en una línea de producto tiene dos retos fundamentales: (1) la presentación de las características comunes y variables de la línea, y (2) la construcción de aplicaciones que incluyan las características comunes, y un subconjunto de las características variables.

Para ello se toma la definición de variabilidad que ofrece Van Group y Bosch, tomada originalmente de (Van Group, y otros, 2002): *"La variabilidad es la habilidad de un sistema software o artefacto para ser cambiado, personalizado o configurado para usarse en múltiples contextos"*.

A partir de lo anterior se define la variabilidad en línea de producto de la siguiente manera: *"La Variabilidad en líneas de productos describe la variación (diferencias) entre los sistemas que pertenecen a una línea de productos en términos de propiedades y cualidades (como características que se proporcionan o requisitos que deben ser cumplidos)."* (Pohl, y otros, 2005). La variabilidad en líneas de productos representa las cualidades variables de los productos de la línea, es decir, los aspectos que pueden cambiar, por ejemplo, el formato de salida de un reporte, en el que las variantes son las posibilidades que existen para guardar ese reporte que pueden ser PDF, DOC y ODT.

Algunos investigadores (Hotz, y otros, 2006) han demostrado que la representación formal de la variabilidad en los artefactos de una LPS es vital para su éxito, donde la forma más común de representar la variabilidad es mediante modelos de características que permiten

además seleccionar la configuración de cada aplicación concreta dentro de una línea de productos.

1.3.1 Variabilidad en la arquitectura

La variabilidad a nivel de arquitectura se consigue mediante la creación de componentes variantes, comunes y opcionales, como por ejemplo, un producto de gama alta puede ofrecer una interfaz de usuario más rica que uno de gama baja. Por tanto, la arquitectura de LPS debería dar cabida a ambas variantes. Para soportar variantes en componentes, hay algunas soluciones típicas, como el diseño por capas (Schmidt, 2000), permitiendo que las diferentes variantes se encapsulen en una capa determinada. También está el patrón arquitectónico Broker (Benavides, y otros, 2007), que puede usarse para reducir el acoplamiento existente entre la parte estable y la parte variable del sistema. También hay una serie de patrones de diseño que se puede usar para gestionar la variabilidad, como la Factoría Abstracta, el patrón Estrategia o el Mediador (Sochos, y otros, 2004).

Hay veces que un componente tiene que ser excluido de un producto. Por ejemplo, hay aplicaciones para las que no es necesario incluir una interfaz de usuario. En estos casos, la arquitectura debe buscar reducir la dependencia con este tipo de componente opcional, aunque se puede seguir accediendo a él cuando esté presente. Hay patrones de diseño que soportan este comportamiento como el patrón arquitectónico Blackboard (Schmidt, 2000). Usando este patrón los componentes opcionales dependen únicamente de la clase blackboard y no de otros elementos del sistema. Otros patrones como los Proxy, que proveen un control de acceso a la clase, junto con el patrón Estrategia permiten a los componentes tener comportamientos diferentes y actuar en contextos diferentes, lo que permitiría la realización de componentes opcionales (Sochos, y otros, 2004).

A partir del concepto de variabilidad planteado anteriormente, esta investigación coincide con el criterio de Van Group al decir que la variabilidad es la habilidad de un sistema para ser cambiado, personalizado o configurado ya que la variabilidad en las líneas de productos representa las cualidades variables de los productos de la línea. El estudio de la variabilidad en la arquitectura de una LPS aportó beneficios significativos a la solución puesto que la variabilidad se consigue mediante la creación de los componentes comunes, variantes y opcionales, siendo útil para la creación de la propuesta.

1.4 Desarrollo basado en componentes

En (Heineman, y otros, 2001; Szyperski, 1998) se brinda una introducción al tema de la Ingeniería de Software Basada en Componentes (ISBC). La ISBC provee metodologías, técnicas y herramientas basadas en el uso y ensamblaje de piezas pre-fabricadas (desarrolladas en momentos diferentes, por distintas personas y posiblemente con distintos objetivos de uso) que puedan formar parte de nuevos sistemas a desarrollar. El desarrollo de software basado en componentes trata de sentar las bases para el diseño y desarrollo de aplicaciones distribuidas basadas en componentes software reutilizables.

A su vez, han surgido nuevos paradigmas que poseen objetivos similares a los de la ISBC, pero que se enfocan en aspectos diferentes a la hora de construir sistemas. Uno de estos paradigmas, fuertemente analizado en la actualidad, es la Ingeniería de Líneas de Productos de Software (ILPS) que intenta construir sistemas mediante el ensamblaje de componentes previamente fabricados. La principal diferencia entre ambos enfoques, es que las líneas de productos son tratadas como un todo, no como productos múltiples que se ven y se mantienen por separado como en el desarrollo de componentes. Sin embargo, a pesar de que el enfoque utilizado en líneas de productos no conlleva al desarrollo de los activos utilizando componentes de software, el uso combinado de ambos enfoques contribuye a reducir el acoplamiento y a incrementar la cohesión, mejorando la modularidad y la evolución de los sistemas construidos (Amin, y otros, 2011; Atkinson, y otros, 2000).

En (Crnkovic, y otros, 2002) se brindan algunos riesgos que se pueden asumir durante el desarrollo basado en componentes, los cuales en las LPS también pueden ocurrir:

- Costo de mantenimiento de los componentes. Aunque los costos de mantenimiento de las soluciones se ven reducidos, lo relativo al mantenimiento de los componentes tiende a elevarse. Esto es así porque, al ser utilizados en varios contextos, los componentes deben responder a una mayor variedad de requisitos y se necesitan emplear más recursos para mantener la trazabilidad, y la adecuación de estos componentes a un universo mayor de consumidores.
- Se requiere tiempo y esfuerzo para desarrollar componentes reutilizables. Como los componentes a crear deben desarrollarse con requisitos no funcionales que aumenten su reutilización, se puede incrementar el tiempo inicial de desarrollo de este tipo de componentes.

1.4.1 Sistemas basados en componentes

Los sistemas de hoy en día son cada vez más complejos, deben ser construidos en tiempo récord y deben cumplir con los estándares más altos de calidad. Para hacer frente a esto, se concibió y perfeccionó lo que hoy conocemos como ISBC, la cual se centra en el diseño y construcción de sistemas computacionales que utilizan componentes de software reutilizables (Sarria Molina, y otros, 2009).

Analizando las definiciones de qué es un componente, por parte de Szyperski, en (Szyperski, 1998) y del Instituto de Ingeniería de Software (SEI, del inglés Software Engineering Institute) en (Brown, 1999), se puede afirmar que un componente, en esencia, es una unidad reutilizable que puede inter-operar con otros módulos software por medio de sus interfaces, las cuales definen desde donde se puede tener acceso a los servicios que este ofrece a los demás componentes.

Un modelo de componentes define la forma de sus interfaces y los mecanismos para interconectarlos entre ellos, por ejemplo, COM, Enterprise JavaBeans o Common Object Request Broker Architecture (CORBA). Basada en un modelo de componentes concreto, una plataforma de componentes es un entorno de desarrollo y de ejecución de componentes que permite aislar la mayor parte de las dificultades conceptuales y técnicas que conlleva la construcción de aplicaciones basadas en los componentes de ese modelo (Krieger, y otros, 1998). En este sentido, podemos definir una plataforma como una implementación de los mecanismos del modelo, junto con una serie de herramientas asociadas.

Sin embargo, el término componente se utiliza a menudo en el sentido más básico de una parte constituyente, elemento, o ingrediente. La plataforma .NET Framework de Microsoft proporciona un soporte para la creación de aplicaciones que utilizan un enfoque basado en componentes. Por ejemplo, esta guía trata sobre los componentes empresariales y de datos, que son comúnmente clases de código compilado en .NET Framework. Ellos ejecutan bajo el control del .NET Framework, y puede haber más de uno de estos componentes en cada ensamblaje.

Microsoft impulsa la idea de industrializar el software utilizando tecnologías de componentes, representando una nueva etapa en la evolución de COM. Las tecnologías COM y CORBA, equivalentes a la era industrial temprana, han logrado producir partes intercambiables básicas. El mayor avance de la Plataforma .NET es la implementación de la parte intercambiable de software en la herramienta de desarrollo misma, de manera que

cada parte de software creada es un componente intercambiable. Este es un avance equivalente a la etapa de línea de ensamblaje de la industrialización.

En (Bosch, y otros, 2002) afirman que la arquitectura y los componentes de un sistema están estrechamente relacionados, y por esta razón, no se puede analizar un sistema basado en componentes sin revisar la arquitectura del mismo. De lo que se infiere que no se puede hablar de los componentes de una LPS sin referirse a su arquitectura.

El disponer de componentes software no es suficiente para desarrollar aplicaciones, ya provengan éstos de un mercado global o sean desarrollados para la aplicación. Un aspecto crítico a la hora de construir sistemas complejos es el diseño de la estructura del sistema, y por ello el estudio de la arquitectura de software se ha convertido en una disciplina de especial relevancia en la ingeniería del software.

1.5 Arquitecturas y estilos arquitectónicos

En este epígrafe se introducen los conceptos de arquitectura y otros relacionados con esta disciplina, además se hace una revisión de los principales estilos arquitectónicos.

1.5.1 Arquitectura de software

Desde el surgimiento de la informática, la programación se consideraba un arte por lo compleja y complicada que era para la mayoría de las personas. Pero poco a poco se comenzaron a desarrollar metodologías para conseguir propósitos y metas. Y a todas estas técnicas se le llamó Arquitectura de Software. La arquitectura de software proporciona una estructura exacta del producto, y es fundamental en el proceso de vida del *software* ya que consiste en un conjunto de patrones, estilos y vistas, entre otros términos que son los que en realidad guían la construcción del sistema. Es la forma en la que los diferentes componentes del sistema se integran para formar un todo unido. Es la forma en la que el sistema encaja en su entorno y con los otros sistemas. La arquitectura se basa específicamente en una serie de objetivos y restricciones funcionales y no funcionales. Es el grado en el que el *software* consigue su propósito fijado y satisface las necesidades del cliente. La arquitectura de software tiene gran importancia y es fundamental para el desarrollo de cualquier sistema informático.

En la actualidad existen un sin número de definiciones de Arquitectura de Software, aunque es válido aclarar que ninguna definición reprocha a la otra, sino que es otra forma de ver la cosas.

Paul Clements define la arquitectura de software como: *"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 la 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 o diferimiento del detalle inherente a la mayor parte de las abstracciones"* (Reynoso).

Otra definición que no deja de ser importante es la definición adoptada por la IEEE en el std. 1471-2000, la cual cita así: "*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).

Por otra parte Bass la define de la siguiente forma: *"La Arquitectura de Software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos."* (Bass, y otros, 1998).

A pesar de la abundante cantidad de definiciones que existen de Arquitectura de Software, la gran mayoría coincide en que esta se refiere a la estructura del sistema, la cual comprende los componentes que forman el sistema, sus interfaces y la comunicación entre ellos. Es muy importante dentro de la arquitectura de software el estudio de los estilos arquitectónicos, ya que los estilos son previos a la elección de las herramientas y plataforma de desarrollo del sistema.

1.5.2 Estilos arquitectónicos

Los estilos arquitectónicos son de gran importancia y fundamentales en el desarrollo de un software, ya que propone las posibles formas en que el sistema puede estar estructurado. Cada uno de ellos tiene características propias que los distinguen de los demás, donde están compuestos por una serie de elementos y restricciones arquitectónicas. Otros autores utilizan el término patrón arquitectónico para definir un concepto muy similar. Si bien ambos no son intercambiables del todo, en ambos bandos se observan los mismos patrones/estilos, incluso con idénticos nombres.

Pressman define el estilo arquitectónico de la siguiente forma: *"Cada estilo describe una categoría del sistema que contiene: un conjunto de componentes por ejemplo, una base de datos, módulos computacionales que realizan una función requerida por el sistema; un*

conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes; restricciones que definen cómo se pueden integrar los componentes que forman 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" (Pressman, 2005).

Mary Shaw y Paul Clements en (Shaw, y otros, 1996) identifican los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. Los componentes, se pueden distinguir por la naturaleza de su computación: por ejemplo, si retienen estado entre una invocación y otra, y de ser así, si ese estado es público para otros componentes. Los tipos de componentes también se pueden distinguir conforme a su forma de empaquetado, o dicho de otro modo, de acuerdo con las formas en que interactúan con otros componentes. El empaquetado es usualmente implícito, lo que tiende a ocultar importantes propiedades de los componentes. Para clarificar las abstracciones se aísla la definición de esas interacciones bajo la forma de conectores: por ejemplo, los procesos interactúan por medio de protocolos de transferencia de mensajes, o por flujo de datos a través de tuberías (*pipes*). Es en gran medida la interacción entre los componentes, mediados por conectores, lo que confiere a los distintos estilos sus características distintivas.

En (Garlan, y otros, 1996), David Garlan, Andrew Kompanek, Ralph Melton y Robert Monroe, definen el estilo como una entidad consistente en cuatro elementos:

1. Un vocabulario de elementos de diseño: componentes y conectores tales como tuberías, filtros, clientes, servidores, bases de datos, etcétera.
2. Reglas de diseño o restricciones que determinan las composiciones permitidas de esos elementos.
3. Una interpretación semántica que proporciona significados precisos a las composiciones.
4. Análisis susceptibles de practicarse sobre los sistemas construidos en un estilo, por ejemplo análisis de disponibilidad para estilos basados en procesamiento en tiempo real, o detección de abrazos mortales para modelos cliente-servidor.

En (Garlan, 2003) un estilo arquitectónico define un vocabulario de tipos para componentes, conectores, interfaces y propiedades, además de un conjunto de reglas que rigen cómo los elementos de dichos tipos pueden componerse. Como se aprecia en la definición anterior se puede describir el estilo arquitectónico como un vocabulario de tipos para componentes, conectores y propiedades, así como reglas que rigen las configuraciones entre ellos. David Garlan en (Garlan, 2003) también plantea el proceso de construcción de un estilo arquitectónico, el cual cuenta de tres pasos:

1. Definición de los elementos que conforman el estilo.
2. Descripción de cada uno de estos elementos.
3. Presentación de las reglas que rigen las combinaciones válidas de los mismos.

Según (Abowd, y otros, 1993) en un estilo es importante distinguir el dominio sintáctico abstracto de la descripción arquitectónica del dominio semántico de significados arquitectónicos.

A partir de la definición anterior, la sintaxis abstracta no está asociada a una arquitectura en específico, sino que define a la arquitectura de software en tres clases abstractas bases: componentes, conectores y configuraciones (Abowd, y otros, 1993). El dominio semántico se centra en describir el significado de la sintaxis abstracta para un estilo específico (Abowd, y otros, 1993).

Además en (Taylor, y otros, 2009) se plantea que un estilo arquitectónico es una colección de decisiones de diseño que son aplicables en un contexto de desarrollo dado, restringen las decisiones de diseño arquitectónicas específicas para un sistema dentro de dicho contexto, y persigue cualidades que benefician los sistemas resultantes.

En la definición anterior, se puede describir el estilo arquitectónico como una colección de decisiones de diseño. Pero no todas las decisiones de diseño caracterizan al estilo arquitectónico, sino que son las decisiones de diseño principales las que lo hacen (Dashofy, 2007).

En esta investigación se considera una buena práctica referirse en un principio a los estilos arquitectónicos clásicos. Por lo que se presenta un resumen de cinco de los estilos arquitectónicos más referenciados actualmente como son: Tuberías y Filtros, Capas, Modelo-Vista-Controlador, Orientada a Objetos y Orientada a Servicios.

Tuberías y Filtros

Este estilo es útil para sistemas que procesan flujos (continuos) de datos, en el que los filtros reciben varios flujos de datos y a su vez producen varios como salida. Las tuberías son las encargadas de transportar estos flujos de datos.

Cada filtro funciona sin tomar en cuenta si los componentes tienen flujo ascendente o descendente; está diseñado para esperar la entrada de datos con cierta forma específica. Sin embargo, no es necesario que el filtro conozca el funcionamiento de los filtros vecinos.

El sistema tubería y filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes.

Para definir este estilo arquitectónico primero hay que definir el elemento Filtro, que constituye el componente de la arquitectura, luego se define las tuberías, que son los conectores y por último se especifica cómo los filtros y las tuberías se combinan para formar un sistema completo.

Arquitectura en capas

En (Shaw, y otros, 1994) definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

Esta arquitectura se caracteriza porque al estar dividido el sistema en capas, las interacciones en la gran mayoría de los casos se realizan entre las capas vecinas, las funcionalidades de cada una de las capas van a estar separadas de las de otras de manera clara, donde cada capa solo contendrá funcionalidades relacionales con las tareas que le corresponde proporcionando un bajo acoplamiento. Se caracteriza también porque las diferentes capas no necesariamente deben estar en una misma máquina sino que pueden estar ubicadas en diferentes niveles físicos.

Los beneficios principales de la Arquitectura por capas son:

- Se pueden entender una sola capa como un todo coherente sin necesariamente conocer sobre las otras capas.
- Mantenibilidad: Provee una organización lógica de aplicación y desarrollo.
- Escalabilidad y rendimiento: Permite distribuir una aplicación, cada capa puede estar separada y alojada en un ordenador distinto, agregar máquinas mejora el rendimiento.

- Seguridad: Permite aislar componentes de tal forma que nunca se tiene acceso a toda la solución.

Es necesario también mencionar las desventajas que puede acarrear el hecho de implementar una Arquitectura en Capas, prejuicios como que las capas no encapsulan bien todas las cosas esto provoca que se obtengan a veces cambios en cascada.

Las capas adicionales pueden cargar la aplicación y disminuir el rendimiento. En cada capa, se debe presentar la información de tal manera que la otra la entienda, lo cual indica un trabajo extra.

Este estilo aporta un diseño basado en niveles de abstracción crecientes, que facilita a los implementadores partir un problema complejo en una secuencia de pasos incrementales. Además admite fácilmente optimizaciones y refinamientos. Sin ignorar que aporta una importante reutilización de los componentes generados.

Al igual que los tipos de datos abstractos, es permisible hacer versiones de una capa siempre y cuando sea viable el uso de las interfaces que la unen con las demás. Esto conduce a la posibilidad de definir interfaces de capa estándar, para a partir de ellas construir extensiones o prestaciones específicas. Estas características hacen que la aplicación cuente con un alto nivel de mutación y escalabilidad permitiéndole al sistema cambios ya sea de requisitos funcionales, agregaciones o mejoras en las funcionalidades con un menor costo de tiempo y esfuerzo.

Modelo-Vista-Controlador

Modelo Vista Controlador (MVC). Es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos (ver Figura 1). El estilo de llamada retorno MVC, se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página.

El modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. No tiene conocimiento específico de los controladores o de las vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el modelo y sus vistas, y notificar a las vistas cuando cambia el modelo.

La vista es el objeto que maneja la presentación visual de los datos representados por el modelo. Genera una representación visual del modelo y muestra los datos al usuario.

El controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del modelo o por alteraciones de la vista.

Este modelo de arquitectura presenta varias ventajas:

- Hay una clara separación entre los componentes de un programa; lo cual permite implementarlos por separado.
- La Interfaz de Programación de Aplicaciones (API siglas en inglés) está bien definida; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.
- La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas.

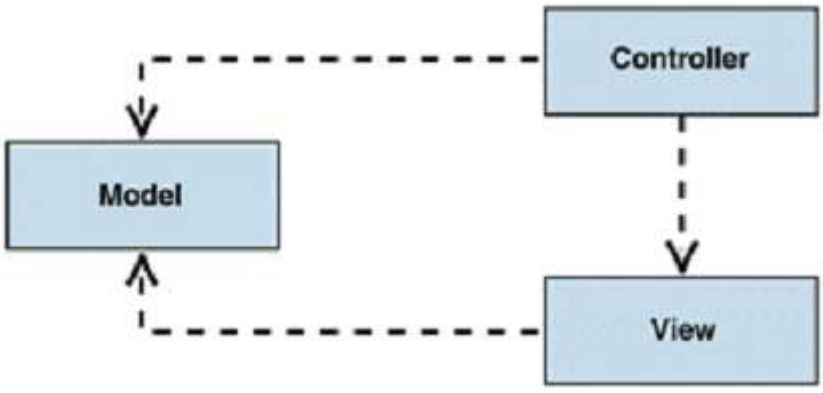


Figura 1 Estilo arquitectónico Modelo-Vista-Controlador. Tomado de (Reynoso, y otros, 2004).

Arquitectura Orientada a Objetos

A este tipo de arquitectura se le conoce de varias maneras, entre ellas: Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos.

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman *managers*, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos (Garlan, y otros, 1994).

Según Billy Reynoso (Reynoso, y otros, 2004), si hubiera que resumir las características de las arquitecturas orientadas a objetos, se podría decir que:

- Los componentes del estilo se basan en principios Orientada a Objetos: encapsulamiento, herencia y polimorfismo. Son así mismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.
- Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología apenas importa si los objetos son locales o remotos. El mejor ejemplo de orientación a objetos para sistemas distribuidos es CORBA, en la cual las interfaces se definen mediante Interface Description Language (IDL); un Object Request Broker media las interacciones entre objetos clientes y objetos servidores en ambientes distribuidos.
- En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. Hay muchas variantes del estilo; algunos sistemas, por ejemplo, admiten que los objetos sean tareas concurrentes; otros permiten que los objetos posean múltiples interfaces.

Ventajas:

- Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- Un objeto es una entidad reutilizable en el entorno de desarrollo.
- Encapsulamiento.

Desventajas:

- Para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.

- Presenta problemas de efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar a A.

Arquitectura Orientada a Servicios

Las Arquitecturas Orientadas a Servicios (SOA: Service Oriented Architecture) están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma y del lenguaje de programación (por ejemplo WSDL[1]). La definición de la interfaz encapsula las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo (como Java o .NET).

Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio desarrollado en CSharp[2] podría ser usado por una aplicación Java.

Una de las características más resaltante de SOA, es que está basada en contratos, donde el proveedor establece las reglas de comunicación, el transporte, y los datos de estrada y salida que serán intercambiados por ambas partes.

SOA es un estilo de arquitectura en el cual se exponen los procesos de negocio del sistema a construir como servicios independientes de alta cohesión y bajo acoplamiento que encapsulan dichos procesos y pueden ser invocados a través de interfaces bien definidas.

Luego de estudiar cada uno de estos estilos arquitectónicos, se puede concluir que la aplicación de uno o varios de estos estilos no es la solución al problema, puesto que son estilos para soluciones más generales. Sin embargo, de ellos se puede tomar ideas, como es el caso del estilo arquitectónico orientado a servicios que una de las cosas que provee es un alto grado de desacoplamiento, aspecto que está relacionado con la mantenibilidad y la reutilización de los productos. El estilo arquitectónico MVC una de las cosas que provee es que las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Esto posibilita que si un componente, posteriormente, se observa que no funciona bien, puede reemplazarse sin que las otras piezas se vean afectadas.

---

[1]Son las siglas de Web Services Description Language, formato que constituye un idioma descriptivo de los servicios web y un estándar del consorcio W3C (World Wide Web Consortium).

[2]Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

Se hizo una búsqueda en *Google Scholar* para encontrar estilos arquitectónico en LPS con las siguientes estrategias de búsqueda (tanto en inglés como español): estilos arquitectónicos, estilos arquitectónicos +LPS, estilos arquitectónicos +Líneas de Productos de Software, architectural styles +SPL, architectural styles +Software Product Lines, entre otros, donde no se encontró ningún resultado. Sin embargo, se encontraron tres compañías (empresas) que cuentan con líneas de productos ERP, como son Estrasol (Estrasol), Calipso (Calipso, 2008) y Microsoft dynamic (Microsoft dynamic). A cada una de estas compañías se le realizó una petición vía correo, donde solamente obtuvimos correos de confirmación diciendo que nos iban a mandar la documentación solicitada pero hasta el momento no hemos recibido otro correo de ellas.

Por lo que se puede concluir que hay muy poca divulgación de este tema, debido a que es difícil encontrar los estilos arquitectónicos empleados en LPS, ya que es un tema de mucha competencia entre todas las empresas, donde ninguna da a conocer sus estrategias para el desarrollo de su línea de productos.

1.6 Conclusiones del capítulo

A lo largo de este capítulo se ha expuesto aspectos a tener en cuenta para seguir con el desarrollo de la investigación, entre ellos se encuentra la concepción de las principales definiciones de LPS y variabilidad. Se analizaron además, la arquitectura de software como activo principal de la LPS, el desarrollo basado en componentes, debido a su generalidad en las LPS. También se analizaron los estilos arquitectónicos, que constituye el mecanismo fundamental para definir muchas de las propiedades de los sistemas. Sin embargo, de ellos se pueden tomar ideas, como es el caso del estilo arquitectónico orientado a servicios que provee es un alto grado de desacoplamiento y el estilo arquitectónico MVC que al construir las piezas de un programa por separado y luego unirlas en tiempo de ejecución posibilita, que si un componente funciona mal puede reemplazarse sin que las otras piezas se vean afectadas.

Capítulo 2: Estilo arquitectónico para la línea de productos

2.1 Introducción

En este capítulo se realiza la propuesta de solución al problema planteado en el diseño teórico de la investigación. Se comienza con una descripción del proceso empleado para la creación del estilo arquitectónico, enfatizando los aspectos que son esenciales para la LPS. Luego se procede a la descripción conceptual del estilo arquitectónico de la línea.

2.2 Desarrollo del estilo arquitectónico

En este epígrafe se presenta el proceso de construcción del estilo arquitectónico de la línea de productos Xedro. La descripción del mismo se realiza siguiendo las propuestas de (Garlan, 2003), pero teniendo en cuenta los puntos de vista de otras fuentes abordas con anterioridad en esta investigación. El estilo arquitectónico se presenta en tres pasos: 1) definición de los elementos que conforman el estilo, 2) descripción de cada uno de estos elementos y 3) presentación de las reglas que rigen las combinaciones válidas de los mismos.

En aras de definir un estilo arquitectónico es imprescindible la definición de los elementos básicos que conforman dicho estilo. Estos elementos son componentes y conectores, los cuales cuentan con una configuración válida en dependencia de las reglas previamente definidas. Las definiciones de componentes y conectores se toman de (Garlan, y otros, 1995).

Componentes

Los componentes son las entidades activas de un sistema computacional (ver Figura 2). Logran tareas a través de la computación interna y la comunicación externa con el resto del sistema. La relación entre un componente y su entorno se define explícitamente como una colección de puntos de interacción, o puertos. Intuitivamente, los puertos generalizan la noción tradicional de una interfaz de módulo. En el caso más simple, un puerto podría representar un procedimiento que se puede llamar o una variable que se puede acceder en una interacción con otro componente. Pero un puerto también podría representar algo mucho más complejo, como una colección de procedimientos, un conjunto de eventos que pueden ser transmitidos o un protocolo de acceso de base de datos.

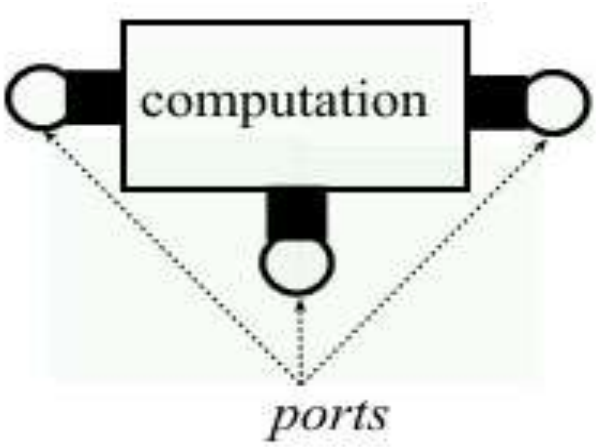


Figura 2 Un componente y un conector. Tomado de (Garlan, y otros, 1995).

Los componentes pueden comunicarse entre sí siguiendo reglas de comunicación determinadas. En la comunicación entre dos componentes siempre debe mediar un conector, en ocasiones este está implícito o condicionado por base tecnológica del sistema.

Conectores

Los conectores definen la interacción entre los componentes (véase la Figura 3). Cada conector proporciona una manera para que una colección de puertos entre en contacto, y define lógicamente el protocolo a través del cual un conjunto de componentes va a interactuar.

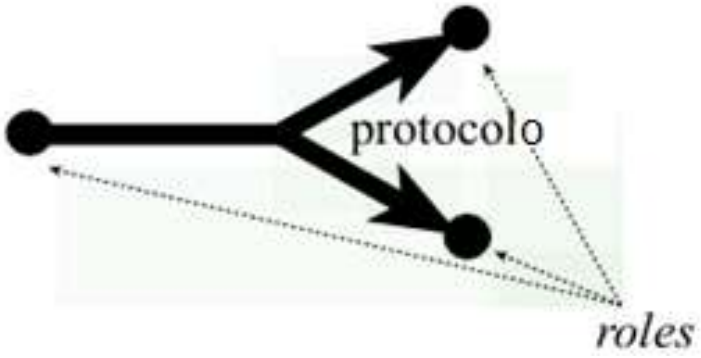


Figura 3 Un conector. Tomado de (Garlan, y otros, 1995).

Al igual que los componentes, los conectores se definen como entidades independientes. Un conector tiene una interfaz que consta de un conjunto de roles. Cada rol define el comportamiento que se espera de uno de los participantes en una interacción. El comportamiento general de un conector (y por lo tanto de la interacción específica) está lógicamente definido por un protocolo. Por lo que la descripción del protocolo de interacción

proporcionada por un conector se separa de su interfaz (de los roles) de la misma manera que la descripción de cálculo de un componente, se separa de su interfaz (de los puertos).

De manera general los conectores se encargan de la integración entre los componentes del sistema. Todos tienen como principal función abstraer las integraciones entre componentes, proporcionando distintos mecanismos en dependencia de los tipos de componentes que intervienen en la comunicación así como el entorno o escenario de comunicación.

2.3 Definición del estilo

Para la definición de los elementos que conforman el estilo arquitectónico, la presente investigación se basa en las definiciones de (Garlan, 2003) y (Taylor, y otros, 2009), las cuales plantean que para esto es necesario describir el estilo a través de un vocabulario de tipos para componentes, conectores y propiedades, así como reglas que rigen las configuraciones entre ellos. Dichas reglas pueden verse como decisiones de diseño, de ahí se define el vocabulario de los componentes y conectores para el estilo, y las decisiones de diseño correspondientes. Para la selección de los atributos que se consideran necesarios se toma como referencia la ontología propuesta por Kruchten en (Kruchten, 2004), la cual se analizó y adaptó para la presente investigación.

2.3.1 Vocabulario de elementos arquitectónicos

En esta investigación se definen dos tipos de componentes principales (contenedores o raíces) que rigen el estilo arquitectónico, en el que cada uno de estos se desglosa en otras categorías de componentes.

El primer tipo de componente son los conceptuales, que contiene los tipos de componentes (Jiménez Méndez, 2000):

- Comunes.
- Variantes.
- Opcionales.

El segundo tipo de componente son los operacionales conteniendo los tipos de componentes (Andrews, y otros, 2002; Matos Arias, y otros, 2013):

- Negocio.
- Dominio.

- Programa.
- Prueba.

Los componentes conceptuales son los encargados de especificar la variabilidad en la arquitectura, es decir determinan en buena medida la variabilidad de los productos en la LPS como se explica en la sección 1.3.1. Además esta investigación se fundamenta en el desarrollo basado en componentes por lo que es necesario definir componentes más específicos de acuerdo a las funciones que tienen estos en el sistema. Estos son los de tipo operacional, es decir son componentes que concretan una operación o procedimiento.

Estos componentes se comunican entre sí mediante conectores. De estos últimos también existen varios tipos en función de los escenarios de comunicación que pueden tener lugar dentro de las configuraciones (Cristiá, 2006), estos son:

- Eventos.
- Llamada a procesamiento.
- IoC.
- Bindig.
- Paso de mensajes.

Cada uno de estos tipos de componentes y conectores define el vocabulario de elementos arquitectónicos para el estilo. En la sección 2.4 se describe cada uno de estos tipos de componentes y conectores para un mejor entendimiento.

2.3.2 Reglas del estilo

En (Matos Arias, y otros, 2013) las reglas del estilo o restricciones representan las reglas sintácticas y semánticas a las que responde cada una de las instancias del estilo arquitectónico. Las restricciones sintácticas definen los tipos de componentes y conectores válidos además de cómo se comunican entre ellos. Por otro lado, las restricciones semánticas pautan el comportamiento interno y externo de los componentes y conectores para una configuración válida.

Según (Taylor, y otros, 2009) se puede describir el estilo arquitectónico como una colección de decisiones de diseño, por lo que las restricciones que se describen en esta investigación corresponde a decisiones de diseño. En (Kruchten, 2004) también se considera la

descripción de la arquitectura de software a partir de un conjunto de decisiones de diseño significativas sobre la organización de un sistema: selección de elementos estructurales y sus interfaces, comportamiento especificado como las colaboraciones entre dichos elementos, las composiciones de los elementos estructurales y de comportamiento dentro de otros subsistemas y el estilo arquitectónico que guía la organización de los anteriores. Además se propone un modelo de decisiones de diseño donde estas son clasificadas, también se sugieren atributos relevantes para cada clasificación así como relaciones entre estas.

Las decisiones de diseño se clasifican en:

- Existenciales, las cuales comprenden las de tipo Estructural y de Comportamiento. Las decisiones estructurales conducen a la creación de subsistemas, capas, divisiones, componentes en algún punto de vista de la arquitectura y las decisiones de comportamiento están más relacionados con cómo los elementos interactúan entre sí para proporcionar una funcionalidad o satisfacer algún requisito no funcional (atributo de calidad), o los conectores.
- Propiedades, que establecen una perdurabilidad, rasgo general o calidad del sistema. Las decisiones de propiedad pueden ser las reglas de diseño o directrices (cuando se expresa positivamente) o restricciones de diseño (cuando se expresa negativamente), como algún rasgo que en el sistema no estará presente.
- Ejecutivas, son las decisiones que no se relacionan directamente con los elementos de diseño o de sus cualidades, sino con el entorno de negocio.

Además de estas clasificaciones es importante almacenar junto con cada decisión, el razonamiento que le dio origen, así como los riesgos de su aplicación. Además de estos atributos se incluirán las relaciones entre decisiones en caso que sean identificadas y un código que servirá para identificarlas más fácilmente.

Las relaciones entre decisiones de diseño pueden ser:

- Restringe (Rg): la decisión B está ligada a la decisión A. Si la decisión A deja de existir, entonces la decisión B también. La decisión B está sujeta a la decisión A, y no se puede promover más alta que la decisión A.

- Prohíbe (Ph): una decisión impide que otra decisión se haga. La decisión objetivo, por lo tanto, no es posible. En otras palabras, la decisión B sólo puede ser promovida a un estado superior a 0, si la decisión A pasa a un estado de 0.
- Permite (Pm): la decisión A hace posible la decisión B, pero no hace que B se tome. También B puede ser decidido aún si A no se toma. Es una forma débil de las restricciones.
- Conflictos con (Cf): una relación simétrica que indica que las dos decisiones de A y B son mutuamente excluyentes.
- Comprende (Cp): la decisión A está hecha de o se descompone en una serie de estrechos, las decisiones de diseño más específicos B1, B2, ..., Bn. Esta relación es más fuerte que la relación restringe, en el sentido de que si el estado de A se degrada, todo el B_i se degrada también. Pero cada B se puede reemplazar individualmente.
- Está relacionada con (Rl): existe una relación de algún tipo entre las dos decisiones de diseño, pero no es de ningún tipo mencionado anteriormente y se mantiene sobre todo por razones de documentación e ilustración.
- Depende (Dp): una decisión de A depende de B si B restringe a A, si B se descompone en A, si A anula B.

2.4 Descripción del estilo

En la sección 2.3.1 se definen los tipos de componentes que se desglosan de los componentes conceptuales y operacionales, teniendo en cuenta la variabilidad en la arquitectura y la función que tienen estos en el sistema. A continuación se describe cada una de estas categorías.

Componentes comunes: son los componentes que se incluyen en la construcción de productos, es decir contienen los procesos que son comunes para toda la línea de productos donde los productos comparten uno o más componentes comunes, se puede utilizar tal y como son o se puede ampliar para habilitar otras funciones.

Componentes variantes: son los componentes que varían o pueden variar en la construcción de productos. Estos componentes presentan los procesos que son variables para toda la línea de productos.

Componentes opcionales: son los componentes que pueden ser parte o no en la construcción de un producto, es decir en estos componentes se encuentran los procesos que pueden estar o no en la línea de productos.

Componentes de negocio: estos componentes están directamente relacionados con funcionalidades de negocio dado que implementan los requerimientos del mismo. Esto quiere decir que el comportamiento de las entidades necesita ser especificado mediante los requerimientos funcionales.

Componentes de dominio: representa en términos generales todos los aspectos del problema del usuario relacionado de forma directa con la funcionalidad que apoya el componente. Estos componentes pueden tener varias funciones en dependencia al tipo de información con que trabajan, pero la forma de recibirla, ofrecerla y procesarla es la misma. Implementa funcionalidades adicionales para el negocio, es decir su función fundamental es permitir configuraciones requeridas para realizar las funcionalidades de negocio.

Componentes de programa: este enfoque es el que más varia de componente a componente, ya que muestra en forma más detallada la información técnica del componente, es decir implementan funcionalidades puramente tecnológicas. Por lo que esta parte del sistema es responsabilidad de un *framework* o marco de trabajo que establece cómo se estructuran los archivos de información, las definiciones de las bases de datos, la definición de las interfaces de datos, información acerca de la invocación de los métodos del componente, etc.

Componentes de prueba: son componentes que funcionan como generadores de datos para probar funciones dentro del sistema con el fin de simular las salidas. Estos componentes tienen un tiempo de vida limitado.

A continuación se describen los tipos de conectores definidos anteriormente:

Eventos: se usan para comunicar los eventos entre los componentes. Existen dos formas de comunicar (1) mediante un bus de eventos, donde cada componente que anuncia un evento lo hace poniendo el evento en el bus. El bus transporta el evento por *broadcast*[3] a todos los componentes; sólo los interesados lo toman y (2) mediante el administrador de eventos, donde este se comunica con el resto de los componentes vía llamada a

[3] Es una forma de transmisión de información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

procedimiento pero esto debería ser transparente para el equipo de desarrollo. Ver anexo # 1.

Llamada a procedimiento: se usan para la comunicación entre dos funciones de un componente, en el que intervienen en comunicaciones intra-componentes. Este tipo de conector tiene como ventaja fundamental permitir una trazabilidad explícita entre las dependencias de los componentes. Por ejemplo, si un componente A requiere un servicio de B, y este, entre las funciones que realiza para dar respuesta a la petición inicial requiere un servicio de un componente C, es necesario explicitar la conexión entre el puerto de salida de B (por el que brinda el servicio a A) y el puerto de entrada de B (por el que consume el servicio de C).

IoC: se usan para mediar en la integración de componentes. Este conector se basa en la implementación del patrón inversión de control e interviene en comunicaciones inter-componentes no agregados entre sí. Este conector puede interpretarse como un tipo de componente de programa.

Bindig: se usan para conexiones entre componentes agregados. Puede ser en ambas direcciones, es decir desde el componente contenedor hacia el agregado o viceversa. Intervienen en comunicaciones inter-componentes agregados entre sí, en ambos sentidos. Este conector indica que la tarea descrita por el puerto del componente que inicia la conexión, será realizada por el componente destino.

Paso de mensajes: se usan en la comunicación de los componentes con el marco de trabajo. Conectan funciones de los componentes entre sí y con las interfaces de servicios ya sean publicados o consumidos.

2.4.1 Descripción de las reglas del estilo

Las reglas del estilo que se presentan a continuación son clasificadas en: Estructurales, de Comportamiento, Restricciones, y Guías. Además las relaciones entre estas reglas son del tipo: comprende (Cp) y está relacionada con (Rl). A partir de lo anterior se muestra el subconjunto de las reglas del estilo.

R01: Los elementos que deben describir el estilo arquitectónico son: componentes y conectores.

Razonamiento: El estilo arquitectónico es descrito mediante componentes y conectores que responden a reglas que rigen las configuraciones válidas de los mismos.

Riesgo: Existen comportamientos del estilo cuya descripción formal puede ser compleja, es decir pueden ser necesarios otros elementos arquitectónicos que por lo general se tienen en cuenta en niveles de abstracción más bajos. Como por ejemplo los patrones de diseño usados durante la implementación.

Categoría: Estructural, Restricción.

Relación: Rl (R02, R03, R04, R05)

R02: Los componentes deben ser de tipo conceptual y operacional.

Razonamiento: El estilo arquitectónico de una línea de productos describe los aspectos comunes y variables de los productos contenidos en la LPS. Para esto es necesario especificar la variabilidad en la arquitectura, es decir los componentes tipo conceptual determinan en buena medida la variabilidad de los productos en la LPS y los componentes operacionales son componentes estratégicos de acuerdo a las funciones que tienen estos en el sistema, es decir son componentes que concretan una operación o procedimiento.

Categoría: Estructural, Restricción.

Relación: Cp (R03, R04)

R03: Los componentes conceptuales agrupan los tipos de componentes comunes, variantes y opcionales.

Razonamiento: Cada una de estas categorías está explicada con anterioridad en el documento.

Categoría: Estructural, Restricción.

Relación: Rl (R01, R02, R04, R05).

R04: Los componentes operacionales agrupan los tipos de componentes de negocio, dominio, programa y prueba.

Razonamiento: Cada una de estas categorías está explicada con anterioridad en el documento.

Categoría: Estructural, Restricción.

Relación: Rl (R01, R02, R03, R05).

R05: Los tipos de conectores del estilo son: eventos, llamada a procedimiento, paso de mensaje, IoC y Bindings.

Razonamiento: Cada uno de estos tipos de conectores determina un rol en un escenario de comunicación dado.

Comunicadores: eventos, llamada a procedimiento, paso de mensaje y IoC.

Coordinadores: Bindings.

Categoría: Estructural, Restricción.

Relación: Rl (R01, R02, R03, R04)

R06: La relación entre los componentes y su entorno es a través de los puertos de entrada y de salida.

Razonamiento: Publican sus funcionalidades a través de interfaces para ofrecer y consumir servicios.

Categoría: Estructural, Restricción.

Relación: Cp (R07)

R07: Los puertos de entrada y los de salida deben ser diferentes en cuanto a su comportamiento.

Razonamiento: Los puertos de entrada representan funcionalidades que proporciona el componente, donde pueden ser accedidas por otros componentes. Mientras que los puertos de salida indican al entorno cuáles son las necesidades del componente para llevar a cabo alguna funcionalidad, es decir los puertos de entrada y de salida se diferencian en cuanto a su comportamiento en el sistema. En caso de que algún componente tenga puertos de entrada o de salida, estos deben quedar explícitos en la descripción de los mismos.

Categoría: Estructural, Restricción.

Relación: Rl (R06, R08, R09)

R08: **Los componentes de programa no poseen puertos de entrada.**

Razonamiento: Los componentes de programa no consumen servicios, es decir estos componentes proveen funcionalidades tecnológicas del sistema, por lo que las funciones que proveen deben tener comportamientos (requisitos no funcionales) específicos.

Categoría: Comportamiento, Guía.

R09: **Los componentes de prueba no poseen puertos de entrada, sino de salida.**

Razonamiento: Los componentes de prueba proveen funcionalidades para probar funciones dentro del sistema, es decir funciona como un generador de datos para simular las salidas.

Categoría: Comportamiento, Guía.

R10: Los componentes deben poseer al menos un puerto de salida para brindar servicios.

Razonamiento: Los componentes que no brinden servicios en el sistema son debido a que no son necesarios en el sistema o a que se hizo un mal diseño del estilo arquitectónico, es decir se debe revisar ya que sus funcionalidades pueden formar parte de otro componente.

Categoría: Comportamiento, Restricción.

R11: Los conectores median la comunicación entre componentes a través de roles.

Razonamiento: Un conector tiene una interfaz que consta de un conjunto de roles. Cada rol define el comportamiento que se espera de uno de los participantes en una interacción. Por lo que en una comunicación entre componentes debe estar bien definido qué rol desempeña cada parte, los roles de los conectores por lo general están implícitos y no se tienen en cuenta en las descripciones arquitectónicas.

Categoría: Comportamiento, Guía.

R12: Los servicios publicados dentro de un componente, contenido dentro de otro, no pueden ser accedidos directamente.

Razonamiento: El acceso debe ser a través del componente que lo contiene sin violar en ningún momento la estructura jerárquica que presentan, es decir los componentes que presentan estructuras jerárquicas no pueden ser accedidos directamente por otros componentes. Si el componente que efectúa el servicio está en el mismo nivel jerárquico que él que lo ofrece y ambos comparten el mismo componente contenedor, es decir el componente raíz, es recomendable permitir dicho acceso.

Riesgo: A la hora de implementar este mecanismo de acceso puede resultar complejo en cuanto al rendimiento, ya que se debe tener registrada la relación jerárquica de todos los componentes. Provocando demoras en los tiempos de respuestas del sistema.

Categoría: Comportamiento, Restricción.

R13: Los servicios publicados dentro de un componente, contenido en otro, no pueden ser accedidos directamente por otros no incluido en dicho componente contenedor.

Razonamiento: El acceso debe realizarse mediante el componente contenedor.

Categoría: Comportamiento, Restricción.

R14: Los componentes que anuncian eventos no deben saber qué otros componentes serán afectados en cada anuncio.

Razonamiento: Cada componente decide si está o no interesado en cada uno de los eventos que son anunciados. En caso de que lo esté, el componente decide cómo reaccionar, en otras palabras ningún componente puede conocer los componentes que serán afectados por el evento que anuncie.

Categoría: Estructural, Restricción.

Relación: Cp (R15)

R15: Los componentes que anuncian eventos pueden comunicarlos a través del bus de eventos y del administrador de eventos.

Razonamiento: Cada una de estas formas de comunicación está explicada con anterioridad en el documento.

Categoría: Estructural, Restricción.

Relación: Rl (R14)

R16: Entre los componentes no deben existir ciclos.

Razonamiento: Un componente puede enviar o recibir eventos hacia o desde los buses a los que está conectado. Los componentes y buses se pueden componer topológicamente de distintas maneras, siguiendo reglas y restricciones particulares, de modo que un componente no puede recibir una notificación generada por él mismo, es decir no se admiten ciclos en los componentes.

Categoría: Comportamiento, Restricción.

R17: Los componentes contenedores deben tener un único tipo de conector IoC cuando es interpretado como un tipo de componente de programa.

Razonamiento: Para la integración de los componentes contenidos en otro se debe hacer a través del conector IoC, cuando es interpretado como un tipo de componente de programa. Dentro del componente contenedor no debe existir otro componente IoC debido a que trae problema a la hora de invocar servicios e integrar los componentes, provocando un alto grado de acoplamiento.

Categoría: Estructural, Restricción.

Relación: Cp (R18)

R18: Los conectores IoC cuando son interpretados como un tipo de componente de programa deben ser de tipo común.

Razonamiento: Revisar R17.

Categoría: Estructural, Restricción.

Relación: Rl (R17)

A continuación se muestra la representación del diagrama de clases que brinda una vista completa del estilo: los conceptos que tiene y sus relaciones.

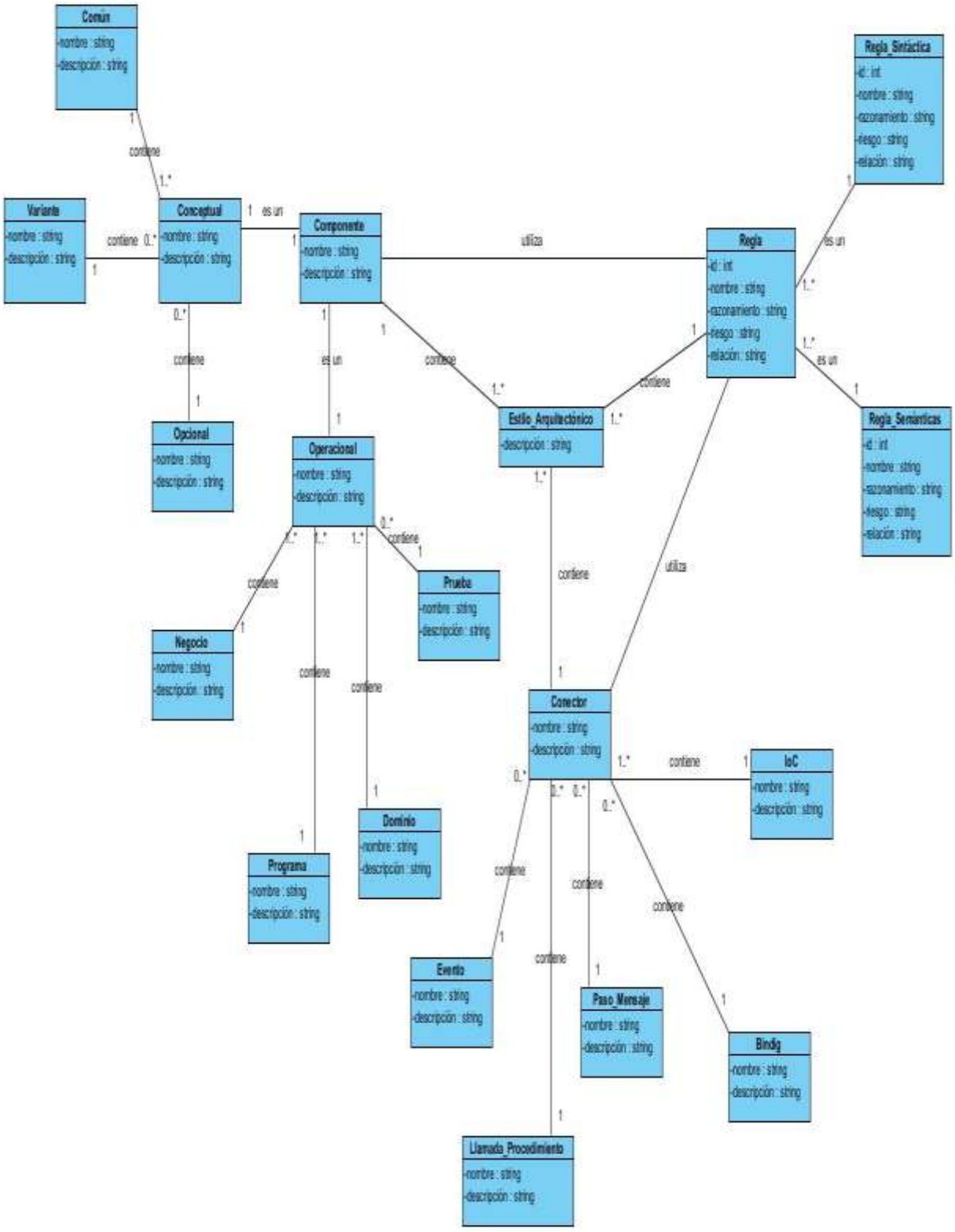


Figura 4 Modelo Conceptual del estilo arquitectónico.

El objetivo de construir un modelo conceptual es conocer, entender, o simular el tema que representan. Este modelo se desarrolla para facilitar el entendimiento del estilo arquitectónico propuesto.

El modelo conceptual propuesto en la solución cuenta con un total de 20 clases, de ellas 3 son clases que representan los tipos de componentes conceptuales encargados de especificar la variabilidad de los productos, 4 de ellas son clases que representan los tipos de componentes operacionales encargados de las funciones que tienen estos en el sistema, es decir son componentes que concretan una operación o procedimiento. Además 5 de estas clases representan los tipos de conectores encargados de integrar los componentes del sistema.

Las otras clases restantes son las encargadas de las reglas del estilo donde las reglas sintácticas definen los tipos de componentes y conectores válidos además de cómo se comunican entre ellos y las reglas semánticas pautan el comportamiento interno y externo de los componentes y conectores para una configuración válida.

2.5 Conclusiones del capítulo

En este capítulo se ha presentado el proceso de construcción del estilo arquitectónico para la línea de productos Xedro, así como la definición y descripción del mismo. Como parte de la construcción del estilo se definieron los tipos de componentes (contenedores) conceptuales y operacionales. Además se definieron los tipos de conectores eventos, llamada a procedimientos, IoC, paso de mensaje y Bindig, que muestran la integración entre los componentes del sistema, así como las reglas que rigen las configuraciones entre ellos.

Capítulo 3: Validación de la solución

3.1 Introducción

En este capítulo se muestran los procedimientos y métodos utilizados para evaluar la calidad de los resultados obtenidos con el desarrollo de este trabajo, haciendo uso de entrevistas a diferentes especialistas para validar las variables dependientes y a través de un catálogo de patrones de variabilidad para validar la propuesta. Es de gran importancia el análisis de los resultados obtenidos ya que esta investigación ayuda a aumentar el nivel de reutilización e integrabilidad de los productos de la LPS Xedro.

3.2 Validación de la solución

El proceso de validación del estilo arquitectónico para garantizar el éxito de la reutilización e integrabilidad de los productos de la línea Xedro del centro CEIGE consta de dos partes. Primero se validan las variables dependientes y luego se valida la variable independiente. Donde las variables a validar son reutilización e integrabilidad (variables dependientes de la investigación) y el estilo arquitectónico (variable independiente de la investigación).

3.2.1 Validación de las variables dependientes de la investigación

La investigación realizada plantea como idea a defender que: con la definición y descripción del estilo arquitectónico se aumenta el nivel de reutilización e integrabilidad de los productos de la línea de productos Xedro. Las variables dependientes de la investigación son: reutilización e integrabilidad de los productos de la línea.

En (Mansell, y otros, 2003; Ruiz, 2010) se plantea que la reutilización e integrabilidad incide sobre la variabilidad de los productos, es decir a mayor nivel de reutilización e integrabilidad se garantiza un mayor grado de variabilidad.

Para la validación se realizaron entrevistas a diferentes especialistas, que se seleccionaron por las siguientes características:

- Graduados de ingeniería en ciencias de informática.
- Experiencia en desarrollo de software mayor de 3 años.
- Haber ocupado el rol de arquitecto o analista (con una experiencia de 3 años como mínimo).
- Posean grados científicos.

- Sean del centro o del departamento de desarrollo de producto.
- Especialistas relacionados con el tema de Arquitectura de Software.

Cada uno de ellos estableció una valoración de los indicadores a evaluar en esta investigación (integrable y reutilizable), donde se ofreció una puntuación del 0 al 10 del cumplimiento del aseguramiento en la propuesta arquitectónica de los indicadores propuestos. Ver Anexo # 2.

Resultados de la evaluación de la métrica

Para la evaluación final de las variables dependientes de la investigación se considera el rango de escala siguiente:

- Entre 1 y 59 puntos porcentuales: se considera que el estilo arquitectónico no cumple con los indicadores a evaluar para aumentar el nivel de reutilización e integrabilidad de los productos de la línea.
- Entre 60 y 79 puntos porcentuales: se considera que el estilo arquitectónico cumple parcialmente con los indicadores a evaluar para aumentar el nivel de reutilización e integrabilidad de los productos de la línea.
- Entre 80 y 100 puntos porcentuales: se considera que el estilo arquitectónico cumple con los indicadores a evaluar para aumentar el nivel de reutilización e integrabilidad de los productos de la línea.

La calidad no tiene sentido sin evaluación, pero en la práctica, en pocos casos se evalúa debido a que evaluar es difícil por: existir atributos de calidad que aún no son bien entendidos, muchas arquitecturas tienen alto nivel de abstracción, descripciones arquitecturales informales y falta de experiencia en cuestiones de evaluación (Astudillo, 2004).

En el caso particular de esta propuesta de estilo arquitectónico los indicadores a evaluar que se propusieron resolver fueron los siguientes (Astudillo, 2004):

Integrable: Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados por separados al ser integrados.

La integrabilidad depende de:

- Complejidad de las componentes.

- Mecanismos y protocolos de comunicación.
- Claridad en la asignación de responsabilidades.
- Calidad y completitud de la especificación de las interfaces.

Los estilos arquitectónicos, son un marco de referencia que se toma en cuenta para construir un sistema. Elegir un estilo arquitectónico es asumir una forma de ver un sistema y tomarlo como referencia para aplicarlo a una realidad y crear una arquitectura. La arquitectura enumerará, en un caso real, los elementos que debe tener un sistema así como la forma en la cual éstos deberán comunicarse y evolucionar. El estilo arquitectónico propuesto asegura la integrabilidad de los componentes ya que al crear diferentes instancias del estilo propuesto y crear la estructura del sistema (la arquitectura), todos los productos a desarrollar toman como referencia el mismo estilo arquitectónico por lo que todos se van a regir por los mismos componentes del sistema, las mismas interfaces y las mismas restricciones de comunicación entre ellos.

Reutilizable: Es la medida de la habilidad del sistema para reutilizar los componentes del sistema, es decir es el proceso de crear sistemas a partir de *software* existente, en lugar de desarrollarlo desde el comienzo.

El desarrollo basado en componentes se apoya en componentes ya desarrollados, que son combinados adecuadamente para satisfacer los requisitos del sistema, logrando un alto grado de reutilización de componentes. El estilo arquitectónico propuesto asegura la reutilización de los componentes ya que tiene en cuenta la variabilidad de los componentes mediante la creación de los componentes comunes, variantes y opcionales, es decir las partes comunes entre los productos se transforman en artefactos de *software* reutilizables, los cuales deben ser creados de forma flexible para que permita añadirle las variabilidades requeridas por el cliente.

Para un mejor análisis de los resultados obtenidos, se hace un análisis estadístico, utilizando como referencia el cálculo de la media aritmética sobre el resultado de los datos obtenidos según la escala propuesta (0-10). A continuación se muestra una representación gráfica de la media de las variables, a partir de los resultados obtenidos de los avales de los especialistas:

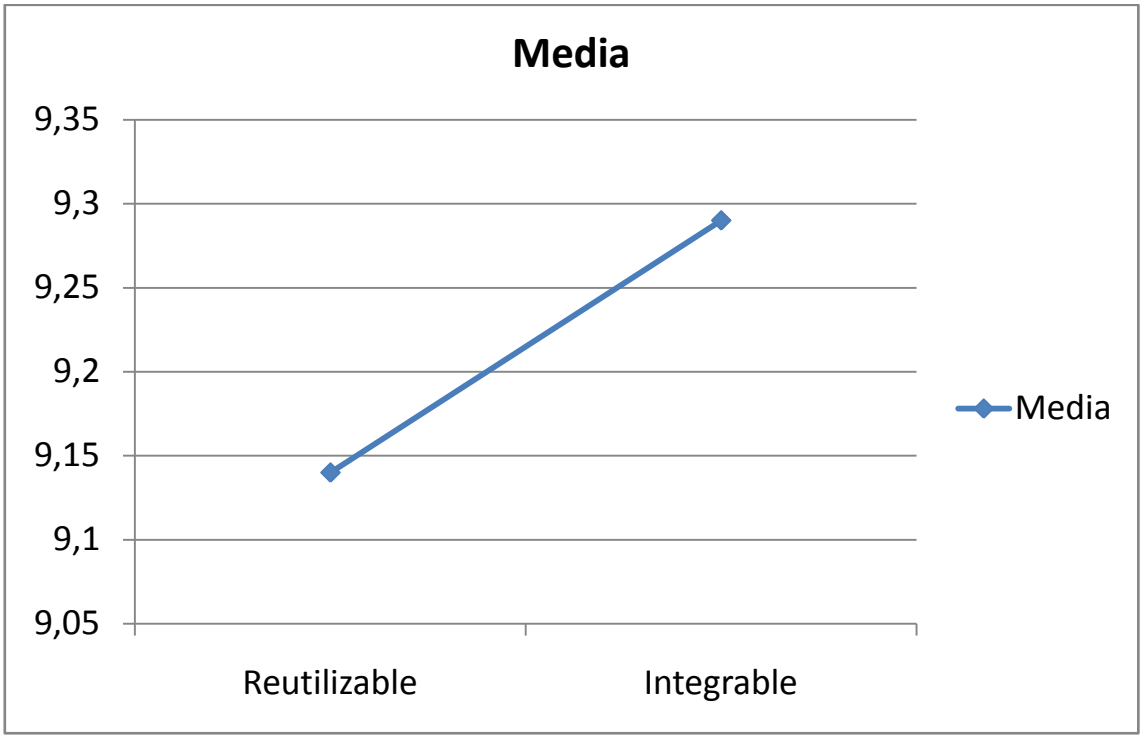


Figura 5 Representación de la media aritmética de los indicadores a evaluar.

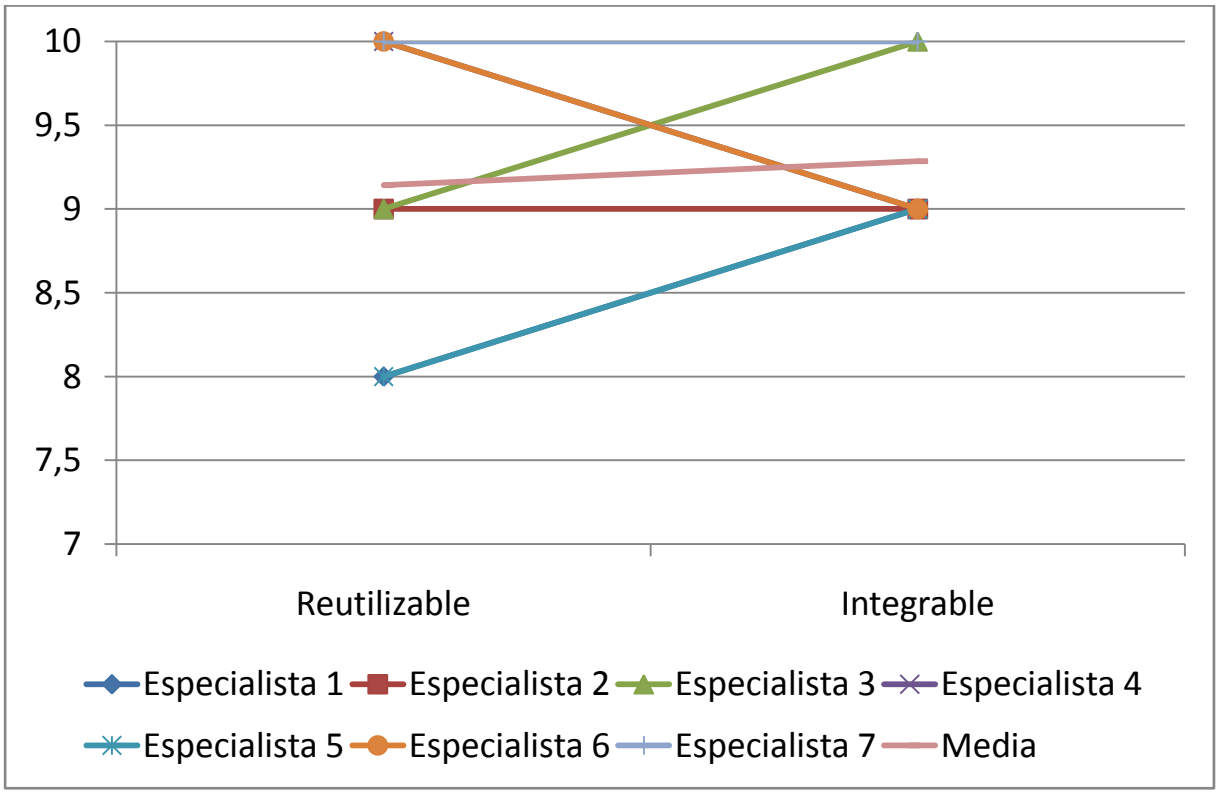


Figura 6 Representación de la media aritmética entre resultados individuales de los indicadores.

Se recogieron opiniones de siete especialistas de la línea de producto arrojando: que cinco profesionales opinaron que el estilo arquitectónico propuesto asegura la integrabilidad de los componentes ya que al crear diferentes instancias del estilo propuesto y crear la estructura del sistema, todos los productos a desarrollar toman como referencia el mismo estilo arquitectónico por lo que todos se van a regir por los mismos componentes del sistema, las mismas interfaces y las mismas restricciones de comunicación entre ellos. Además el estilo arquitectónico propuesto asegura la reutilización de los componentes ya que tiene en cuenta la variabilidad de los componentes mediante la creación de los componentes comunes, variantes y opcionales, es decir se puede definir los puntos reutilizables de la línea. Ofreciendo cada uno de ellos una puntuación de 9 y 10 respectivamente.

Mientras que dos profesionales opinaron que el estilo arquitectónico propuesto asegura la reutilización e integrabilidad de los componentes, pero si la preparación de los desarrolladores no es muy buena entonces no se va a obtener un alto nivel de reutilización de los componentes. Ofreciendo una puntuación de 8 y 9 respectivamente.

Según el análisis de los datos ofrecidos por los profesionales en las entrevistas de acuerdo a la escala propuesta, se establece una media de cómo se cumplen estos indicadores.

Indicadores	Media	%
Reutilizable	9,14	91,4
Integrable	9.29	92.9

Tabla 1 Media de los indicadores evaluados.

3.2.2 Validación de la variable independiente de la investigación

Para la validación del estilo arquitectónico se realizó un catálogo de patrones de variabilidad para la línea Xedro, teniendo en cuenta buenas prácticas para definir el estilo arquitectónico de la línea de productos. A continuación se muestran cada uno de estos patrones a tener en cuenta para la definición del estilo arquitectónico para la línea de productos Xedro y como contribuyeron a la realización del mismo.

Desarrollo basado en componentes

En (Rodríguez Cano, 2002) se plantea que las líneas de productos buscan el desarrollo de componentes que realmente vayan a ser utilizados, es decir, que haya al menos un producto de dicha línea que haga uso de ese componente (lógicamente, podrá ser utilizado en más productos que puedan resultar de la línea de productos). El desarrollo de software basado en componentes se fundamenta en el ensamblaje de componentes pre-elaborados. Este desarrollo de software trae consigo diversos beneficios como:

- Reutilización del software. Nos lleva a alcanzar un mayor nivel de reutilización de software.
- Simplifica las pruebas. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- Mayor calidad. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

El objetivo de este desarrollo basado en componentes es maximizar y también encontrar lo común evitando las duplicidades y compartiendo los recursos, además de controlar e identificar las variaciones estableciendo puntos de control. Un ejemplo ilustrativo de esto se muestra en la figura 7.

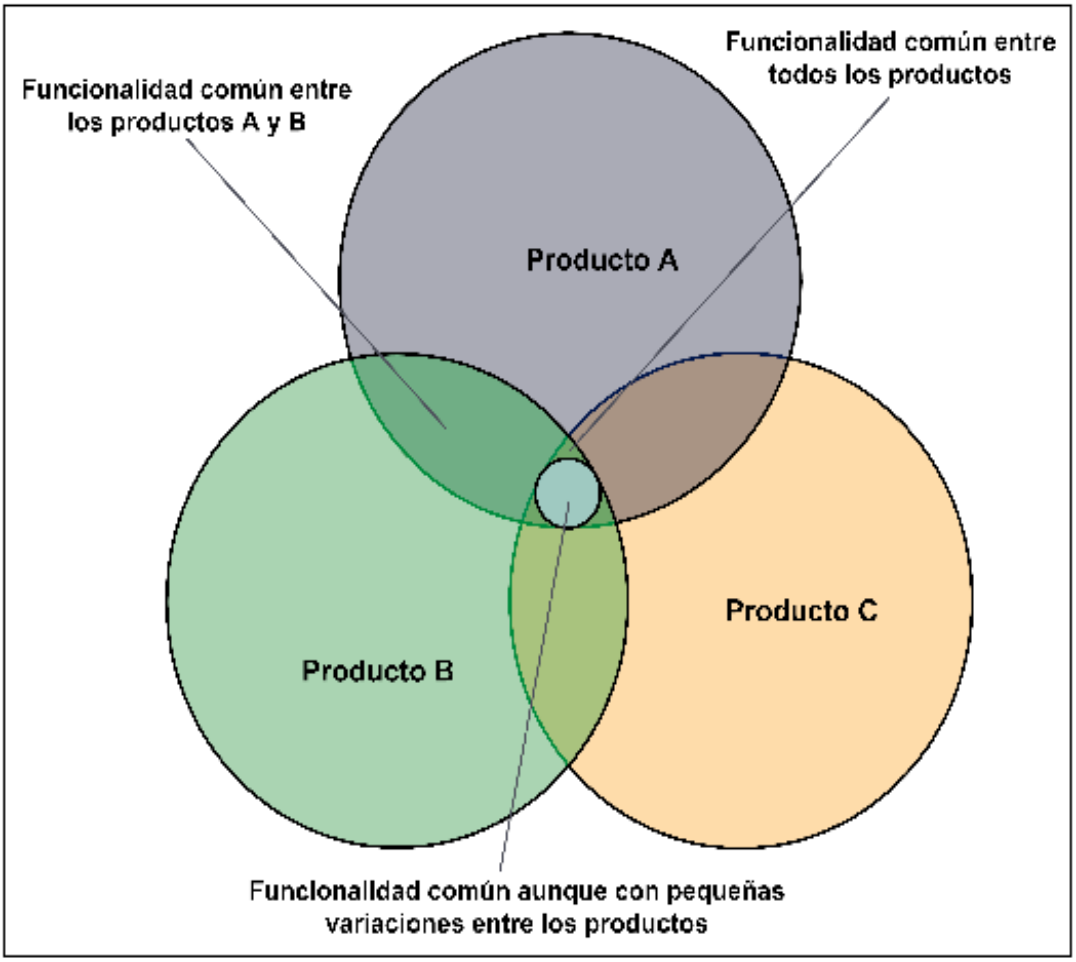


Figura 7 Diagrama de variabilidad y elementos comunes entre productos. Tomado de (Rodríguez Cano, 2002).

El estilo arquitectónico propuesto se definió a partir del desarrollo basado en componentes ya que se basa en el desarrollo de componentes reutilizables, aunque no es en sí mismo un modelo de desarrollo de software, utiliza el mismo principio básico que las LPS.

Componentes y Conectores

David Garlan en (Garlan, 2003) plantea que para la construcción de un estilo arquitectónico, es necesario la definición de los elementos que conforman el estilo, la descripción de cada uno de estos elementos y por último la presentación de las reglas que rigen las combinaciones válidas de los mismos.

En aras de definir un estilo arquitectónico es imprescindible la definición de los elementos básicos que conforman dicho estilo. Estos elementos son los componentes y los conectores, los cuales cuentan con una configuración válida en dependencia de las reglas previamente

definidas. A partir de lo anterior el estilo arquitectónico propuesto en esta investigación se definió de acuerdo a estos elementos arquitectónicos, en el cual los componentes son las entidades activas de un sistema computacional y pueden comunicarse entre sí siguiendo reglas de comunicación determinadas. Además en la comunicación entre dos componentes siempre debe mediar un conector y los conectores se encargan de la integración entre los componentes del sistema.

Componentes comunes, variantes y opcionales

La arquitectura de una LPS captura los aspectos comunes y variables de una línea de productos, en el cual los aspectos comunes de la arquitectura son capturados por los componentes de software que son comunes a toda la línea y los aspectos variables de la arquitectura son capturados por los componentes de software que varían entre los miembros de la línea. Para permitir la construcción de distintos productos en una línea, debe definirse una serie de puntos de variación que son necesarios para poder crear los distintos productos. La arquitectura también debe asegurar que existan mecanismos de variabilidad en los productos (Krueger, 2007).

En (Jiménez Méndez, 2000), Alberto Jiménez M. plantea que para lograr una mayor variabilidad en una línea de producto hay que tener en cuenta los aspectos variables de la misma. Donde se consigue mediante la creación de componentes variantes, comunes y opcionales. A partir de lo anterior el estilo arquitectónico propuesto en esta investigación se definió de acuerdo a estas variantes en componentes, en el que para una LPS es esencial representar la variabilidad de los componentes. Teniendo en cuenta que se quiere lograr un mayor grado de variabilidad para la línea de productos Xedro.

Reglas del estilo

Según Garlan en (Garlan, 2003) para la definición de un estilo arquitectónico es necesario la representación de las reglas que rigen las combinaciones válidas de los mismos. Según (Taylor, y otros, 2009) se puede describir el estilo arquitectónico como una colección de decisiones de diseño, por lo que las restricciones que se describen en esta investigación corresponden a decisiones de diseño. A partir de lo anterior las reglas del estilo arquitectónico propuesto en esta investigación se definieron de acuerdo a las definiciones de Garlan y Taylor, teniendo en cuenta la ontología propuesta por Kruchten en (Kruchten, 2004).

3.3 Resultados obtenidos

Los resultados obtenidos de la aplicación de la métrica para validar las variables dependientes arroja como evaluación general en términos porcentuales un 92 % que en la escala propuesta significa que el estilo arquitectónico cumple con los indicadores a evaluar para aumentar el nivel de reutilización e integrabilidad de los productos de la línea. A continuación se muestra una representación gráfica de la evaluación de la métrica, a partir de los resultados obtenidos de las entrevistas de los especialistas:

Figura 8 Representación en porciento (%) de la evaluación de la métrica.

A partir del análisis realizado anteriormente, se considera que el estilo arquitectónico propuesto: responde a los mecanismos de variabilidad de la línea de productos, aumenta el nivel de reutilización e integrabilidad de los productos de la línea. Además se tuvo en cuenta buenas prácticas a la hora de definir y describir el estilo arquitectónico propuesto ya que se tomaron las definiciones de Garlan y Taylor en (Garlan, 2003) y (Taylor, y otros, 2009) respectivamente.

3.4 Premios otorgados a la solución

Como mecanismo para constatar la opinión de diversos especialistas en la materia y de la comunidad científica de manera general se presentó el trabajo en diversos eventos científicos. Los resultados obtenidos demuestran la novedad y pertinencia del tema, además las observaciones realizadas contribuyeron a elevar la calidad de la propuesta.

La presente investigación fue presentada en varios eventos los cuales son mencionados a continuación:

- El 9 de mayo se le otorgó el premio de Destacado en la XII Jornada Científica Estudiantil a nivel de Facultad.
- El 22 de mayo se le otorgó el premio de Relevante en la XII Jornada Científica Estudiantil a nivel de Universidad.

Para más detalle (Ver anexo 3) y (Ver anexo 4) respectivamente.

3.5 Conclusiones del capítulo

Durante este capítulo se proponen un conjunto de acciones con el objetivo general de garantizar la calidad del estilo arquitectónico propuesto. Se analizan los resultados de las entrevistas realizadas y de las encuestas, reflejando los indicadores medidos y los resultados, mediante gráficos de pastel y porcientos. La validación de la variable independiente proporcionó una medida cualitativa de la calidad del estilo arquitectónico. Por otra parte, la validación de las variables dependientes permitió obtener, de forma general, una medida cuantitativa (porciento) de acuerdo a los indicadores evaluados, donde posibilitó el cumplimiento de los mismos para aumentar el nivel de reutilización e integrabilidad de los productos de la línea.

Conclusiones generales

Como resultado de la investigación realizada en el presente trabajo se arribaron a las siguientes conclusiones:

1. La definición del marco teórico permite una mayor comprensión del contexto donde se realiza la investigación, a partir del mismo se define los enfoques actuales de la arquitectura, principales tendencias y definiciones apropiadas para contribuir el estilo arquitectónico de la línea de productos Xedro.
2. Se analizaron los patrones y tácticas de variabilidad referente a una línea de productos permitiendo definir y describir el estilo arquitectónico propuesto para la línea de productos Xedro.
3. Se validaron las variables que forman parte del problema de la investigación y se demuestra que el estilo arquitectónico contribuye a elevar el nivel de reutilización e integrabilidad de los productos de la línea de productos Xedro.

Recomendaciones

Después de haber analizado las conclusiones a las que se llegan con la realización de esta investigación, se recomienda:

- Evaluar el estilo arquitectónico propuesto a partir de métodos de evaluación de calidad para ver cómo se comporta ante distintos escenarios que respondan a los atributos de calidad: seguridad, portabilidad y escalabilidad; y de esta forma identificar sus debilidades.
- Se recomienda aplicar el estilo arquitectónico en un contexto real.

Bibliografía referenciada

Abowd, G., Allen, R. y Garlan, D. 1993. *Using Style to Understand Descriptions of Software Architecture.* California : Proceedings of the ACM SIGSOFT'93 Symposium on Foundations of Software Engineering, Redondo Beach, 1993.

Amin, F., Mahmood, A. K. y Oxley, A. 2011. *Reusability Assessment of Open Source Components for Software Product Lines.* s.l. : International Journal of New Computer Architectures and their Applications (IJNCAA), 2011.

Andrews, Anneliese y Sudipto, Ghosh. 2002. *A model for Understandig Software Componets.* 2002.

Astudillo, Hernán. 2004. Arquitectura de Software Recuperación y evaluación. [En línea] portal.inf.utfsm.cl, 2004. [Citado el: 06 de 06 de 2014.] http://www.inf.utfsm.cl/~hernan/cursos/INF326-2004s1/bitacora/sesion_16.pdf.

Atkinson, C., Bayer, J. y Muthig, D. 2000. *Component-based product line development: The kobra approach.* 2000. ISBN 576:289-309.

Bass, L., Clements, P. y Kazman, R. 1998. *Sofhlare Architecture in Practice.* s.l. : Addison-Wesley, 1998.

Benavides, D., y otros. 2007. *FAMA: Tooling a framework for the automated analysis of feature models.* Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS) : s.n., 2007. págs. 129–134.

Bosch, J. 2000. *Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach.* s.l. : Addison-Wesley, 2000.

Bosch, Jan y Stafford, Judith A. 2002. *Architecting component-based systems.* 2002. ISBN 1-58053-327-2.

Brown, A. 1999. *Constructing Superior Software, capítulo Building Systems from Pieces: Principles and Practice of Component-based Software Engineering.* s.l. : Macmillan Technical Publishing, 1999. ISBN: 15-787-01473..

Calipso. 2008. Calipso. [En línea] 2008. [Citado el: 10 de 03 de 2014.] http://www.calipso.com/.

Casallas, Rubby. *Líneas de productos de software - Introducción.* Bogotá : s.n. Presentación.

Clements, Paul y Northrop, Linda. 2001. *Software Product Lines: Practices and Patterns.* s.l. : SEI Series en Software Engineering. Addison–Wesley, 2001.

—. 2002. *Software Product Lines: Practices and Patterns.* s.l. : Addison-Wesley, 2002.

Clements, Paul. C. 1996. *A Survey of Architecture Description Languages.* s.l. : 10, 1996.

Cristiá, Maximiliano. 2006. *Catálogo Incompleto de Estilos Arquitectónicos.* Universidad Nacional de Rosario : s.n., 2006.

Crnkovic, Ivica y Larsson, Magnus. 2002. *Building Reliable Component-Based Software Systems.* Artech House, Norwood, MA : s.n., 2002. ISBN 1-58053-327-2.

Dashofy, E. M. 2007. *Supporting Stakeholder-driven, Multi-view Software Architecture Modeling.* Tesis doctoral, Universidad de California, Irvine : s.n., 2007.

Díaz, Oscar y Trujillo, Salva. 2010. *Líneas de productos de software.* [ed.] M. G. Piattini y J. Garzás. Segunda edición. s.l. : Fábricas de Software: experiencias, tecnologías y organización, 2010.

Estrasol. Estrasol: enabling the strategy. [En línea] [Citado el: 10 de 03 de 2014.] http://www.estrasol.com.mx/.

Gamma, E., y otros. 1995. *Design patterns: elements of reusable object-oriented software.* s.l. : Addison-Wesley Professional, 1995.

Garlan, David. 2003. *"Formal Modeling and Analysis of Software Architecture: Components, Connectors, and Events".* [ed.] Lecture Notes in Computer Science. s.l. : Formal Methods for Software Architectures, 2003. págs. 1-24. Vol. 2804.

Garlan, David y Shaw, Mary. 1994. *An introduction to software architecture.* s.l. : CMU Software Engineering Institute Technical Report, 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.

Garlan, David, Abowd, Gregory y Allen, Robert. 1995. *Formalizing Style to Understand Descriptions of Software Architecture.* 1995.

Garlan, David, y otros. 1996. *"Architectural Style: An Object-Oriented Approach".* 1996.

Heineman, G. T. y Councill, W. T. 2001. *Component-based software engineering: putting the pieces together.* Boston, MA, USA : Addison-Wesley Longman Publishing Co., 2001. Inc..

Hotz, L., y otros. 2006. *Configuration in Industrial Product Families.* s.l. : The ConIPF Methodology. IOS Press, 2006. ISBN 978-1-58603-641-6.

Jiménez Méndez, Alberto. 2000. *Gestión de la Variabilidad en Líneas de Producto Software.* Departamento de Lenguajes y Sistemas Informáticos, E.T.S.I.I. - Universidad de Sevilla : s.n., 2000.

KIM, W. C. y Mauborgne, R. 2005. *Blue Ocean Strategy.* Boston. Harvard Business School : s.n., 2005.

Krieger, D. y Adler, R. M. 1998. *The Emergence of Distributed Component Platforms.* Computer Journal : s.n., 1998. págs. 41(3):43–53.

Kruchten, P. 2004. *"An Ontology of Architectural Design Decisions in Software-Intensive Systems".* [ed.] Jan Bosch. s.l. : Proc. of the 2nd Workshop on Software Variability Management, Groningen, NL, 2004. 3-4.

Krueger, Charles W. 2007. *The 3-tiered methodology: Pragmatic insights from new generation software product lines.* Software Product Line Conference, International : s.n., 2007. págs. 97-106.

Mansell, Jason y Maiza, Mikel. 2003. *La variabilidad en el Software ¿cómo identificarla y gestionarla para conseguir mayor eficiencia en el desarrollo?* 2003.

Matos Arias, Y. y Silega Martínez, N. 2013. *Estilo arquitectónico para el sistema integrado de gestión Cedrux.* s.l. : GECONTEC: Revista Internacional de Gestión del Conocimiento y la Tecnología, 2013. Vol. 1.

Microsoft dynamic. Microsoft dynamics. [En línea] [Citado el: 10 de 03 de 2014.] http://www.microsoft.com/es-xl/dynamics/erp.aspx.

Montilva, Jonás A. 2006. *Desarrollo de Software Basado en Líneas de Productos de Software.* 2006. Presentación.

Oliveira, E.A.d., y otros. 2005. *A variability management process for software product lines.* en Conference of the Centre for Advanced Studies on Collaborative research Toronto, Ontario, Canada : s.n., 2005. págs. 225 - 241.

Pohl, K., Böckle, G. y Linden, F. v. d. 2005. *Software Product Line Engineering. Foundations, Principles and Techniques.* Berlin Heidelberg: Springer : s.n., 2005.

Pohl, Klaus y Halmans, Günter. 2003. *Communicating the variability of a software-product family to customers.* s.l. : Software and System Modeling, 2003. págs. 15-36. 2(1).

Pressman, Roger S. 2005. *Ingeniería de Software. Un Enfoque Práctico.* La Habana Cuba : Félix Varela, 2005.

Reynoso, Carlos Billy y Kicillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Universidad de Buenos Aires : s.n., 2004.

Reynoso, Carlos. *Introducción a la Arquitectura de Software.* UNIVERSIDAD DE BUENOS AIRES : s.n.

Rodríguez Cano, Guillermo. 2002. *Líneas de productos y modelos de características. Un caso de estudio en el asociacionismo.* Universidad de Valladolid. E. T. S. DE INGENIERÍA I NFORMÁTICA. Ingeniería Técnica en Informática de Sistemas : s.n., 2002.

Ruiz, Francisco. 2010. *Calidad de Procesos y Productos Software.* Universidad de Cantabria : s.n., 2010.

Sarria Molina, Arledy y Díaz Ariza, Nazly Johana. 2009. *EXPOSICIÓN SOFTWARE POR COMPONENTES.* Universidad del valle. Escuela de Ingeniería de Sistemas y Computación. Tecnología en Sistemas de Información : s.n., 2009.

Schmidt, D.C., Rohnert, H., Stal, M., Schultz, D. 2000. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects.* New York, NY, USA : John Wiley & Sons, Inc., 2000.

Shaw, Mary y Clements, Paul. 1996. *"A field guide to Boxology: Preliminary classification of architectural styles for software systems".* Documento de Computer Science Department and Software Engineering Institute, Carnegie Mellon University : Publicado en Proceedings of the 21st International Computer Software and Applications Conference, 1996.

Shaw, Mary y Garlan, David. 1994. *An introduction to software architecture.* s.l. : CMU Software Engineering Institute Technical Report, 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.

Sochos, P., Philippow, I. y Riebisch, M. 2004. *Feature-oriented development of software product lines: Mapping feature models to the architecture.* Object-Oriented and Internet-Based Technologies : s.n., 2004. págs. 138–152.

Szyperski, Clemens A. 1998. *Component software - beyond object-oriented programming.* s.l. : Addison-Wesley-Longman, 1998.

Szyperski, Clemens. 1998. *Component Software. Beyond Object-Oriented Programming.* s.l. : Addison-Wesley, 1998. ISBN: 0-201-17888-5.

Taylor, R., Medvidovic, N. y Dashofy, E. 2009. *Software architecture: Foundations, theory and practice.* 2009.

Van Group, J. y Bosch, J. 2002. *Design Erosion: Problems and Causes, Journal of Systems & Software.* 2002.

Bibliografía consultada

Abowd, Gregory, Allen, Robert y Garlan, David. 1995. *Formalizing Style to Understand Descriptions of Software Architecture.* Carnegie Mellon University. Pittsburgh, PA 15213 : Computer Science Department, 1995.

Ali Babar, Muhammad, Chen, Lianping y Shull, Forrest. 2010. *Managing Variability in Software Product Lines.* 2010.

Allen, Robert y Garlan, David. 1992. *Towards Formalized Software Architectures.* Carnegie Mellon University. Pittsburgh, PA 15213 : School of Computer Science , 1992. CMU-CS-92-163.

Bachmann, Felix y Clements, Paul C. 2005. *Variability in Software Product Lines.* Carnegie Mellon University. Pittsburgh, PA 15213-3890 : Product Line Practice Initiative, 2005. TECHNICAL REPORT. CMU/SEI-2005-TR-012. ESC-TR-2005-012.

Bartholdt, Joerg, Oberhauser, Roy y Rytina, Andreas. 2008. *An Approach to Addressing Entity Model Variability within Software Product Lines.* 2008. ISBN 978-0-7695-3372-8/08.

Bastarrica, María Cecilia. *Desarrollo de Líneas de Productos de Software.* Universidad de Chile : s.n.

Bosch, Jan, y otros. *Variability Issues in Software Product Lines.*

Chen, Lianping y Ali Babar, Muhammad. 2011. *A systematic review of evaluation of variability management approaches in software product lines.* 2011.

—. 2010. *A systematic review of evaluation of variability management approaches in software product lines.* 2010.

Clements, Paul y Northrop, Linda. *Software Product Lines.* Carnegie Mellon University. Pittsburgh, PA 15213 : Product Line Systems Program.

Cortés Verdín, M. K., Lemus Olalde, C. y Montes de Oca Vázquez, C. 2008. *Enfoque Orientado a Aspectos para una Arquitectura de Línea de Productos de Software.* Cuernavaca, Morelos., México : s.n., 2008.

García Peñalvo, Francisco José, y otros. 2002. *Líneas de Productos, Componentes, Frameworks y Mecanos.* Departamento de Informática y Automática - Universidad de Salamanca : s.n., 2002. CICYT TIC2000-1673-C06-05.

Garlan, David, Monroe, Robert T . y Wile, David. 1997. *ACME: An Architecture Description Interchange Language.* s.l. : Tepper School of Business, 1997.

González Rodriguez, Belkis Grissel, y otros. 2012. *Propuesta de requisitos básicos para la línea de productos de software Scada del Centro de Informática Industrial.* Serie Científica de la Universidad de las Ciencias Informáticas. La Habana. Cuba : s.n., 2012. Vol. 5.

Hernández, Ocharán, y otros. *Documentando Arquitecturas Orientadas a Aspectos para Líneas de Productos de Software.*

Heymans, Patrick y Trigaux, Jean-Christophe. 2003. *Software Product Lines: State of the art.* s.l. : Product Line ENgineering of food TraceabilitY software, 2003. EPH3310300R0462 / 215315.

Laguna, Miguel A. 2009. *Desarrollo de Líneas de Productos: un Caso de Estudio en Comercio Electrónico.* San Cristóbal, Venezuela : s.n., 2009.

Laguna, Miguel A., González Baixauli, Bruno y López, Oscar. *Gestión de la Variabilidad en Líneas de Productos.*

Lin, Yuqing, Ye, Huilin y Tang, Jianmin. 2009. *Measurement of the Complexity of Variation Points in Software Product Lines.* Australia : s.n., 2009. ISBN 978-0-7695-3570-8/09.

Mansell, Jason y Maiza, Mikel. 2003. *La variabilidad en el Software ¿cómo identificarla y gestionarla para conseguir mayor eficiencia en el desarrollo?* 2003.

Mellado, Daniel, Fernández Medina, Eduardo y Piattini, Mario. 2010. *Security requirements engineering framework for software product lines.* 2010.

Midwinter, Andrew, y otros. *Software Architectural Styles.*

Monroe, Robert T., y otros. 1996. *Architectural Styles, Design Patterns, and Objects.* Carnegie Mellon University : Tepper School of Business, 1996.

Muro Fumero, Dorisbel, Lazo Ochoa, René y González Dum, Roberto. 2011. *Modelo de desarrollo de software basado en Líneas de Producción de Software para la industria cubana.* Serie Científica de la Universidad de las Ciencias Informáticas. La Habana. Cuba : s.n., 2011. Vol. 4.

Osman Elfaki, Abdelrahman, Phon Amnuaisuk, Somnuk y Kuan Ho, Chin. 2009. *Modeling Variability in Software Product Line using First Order Logic.* 2009. ISBN 978-0-7695-3903-4/09.

Peng, Xin, Yijun, Yu y Zhao, Wenyun. 2011. *Analyzing evolution of variability in a software product line: From contexts and requirements to features.* 2011.

Ribeiro, Márcio y Borba, Paulo. 2009. *Improving Guidance when Restructuring Variabilities in Software Product Lines.* Brazil : s.n., 2009. ISBN 1534-5351/09.

Schmerl, Bradley y Garlan, David. 2003. *AcmeStudio: Supporting Style-Centered Architecture Development.* Carnegie Mellon University. 5000 Forbes Ave, Pittsburgh, PA 15213 : s.n., 2003.

Sinnema, Marco, y otros. *COVAMOF: A Framework for Modeling Variability in Software Product Families.* Department of Mathematics and Computing Science, University of Groningen, PO Box 800, 9700 AV Groningen, The Netherlands : s.n.

Taylor, Richard N., Medvidovic, Nenad y Oreizy, Peyman. *Architectural Styles for Runtime Software Adaptation.*

Taylor, Richard N., y otros. 1996. *A Component- and Message-Based Architectural Style for GUI Software.* s.l. : IEEE Transactions on Software Engineering, 1996. págs. 390-406. Vol. 22.

Van Gurp, Jilles, Bosch, Jan y Svahnberg, Mikael. *Managing Variability in Software Product Lines.* University of Groningen : s.n.

Glosario de términos

API: Aplication Programming Interface – Interfaz de Programación de Aplicaciones, es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizada por otro software.

Componente: Un componente de software es una pieza reusable de código y datos en forma binaria que se puede conectar a otros componentes software de otros fabricantes con un esfuerzo relativamente pequeño (Brockschmidt, 1993). Son independientes entre ellos, y tienen su propia estructura e implementación.

CORBA: Common Object Request Broker Architecture – arquitectura común de intermediarios en peticiones a objetos, es un estándar que establece una plataforma de desarrollo en ambientes distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

CSharp: Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

Framework: Marco de trabajo, solución reutilizable y extensible que permite crear componentes propios a partir de los que el provee.

IEEE: Son las siglas en inglés de The Institute of Electrical and Electronics Engineers (El Instituto de Ingenieros Eléctricos y Electrónicos), una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

IoC: Son las siglas de Inversion of Control (Inversión de Control). Mueve la responsabilidad de realizar las tareas al inferior del framework y fuera del código de la aplicación.

Java: Tecnología, lenguaje de programación creado por la Sun Microsystems.

MVC: Son las siglas en inglés de Model View Controller (Model Vista Controlador). Es un patrón que consiste en separar la interfaz de usuario, los datos y la lógica de control en tres componentes distintos.

Objeto: En el paradigma de la Programación Orientada a Objetos (POO, o OOP en inglés), un objeto es la unidad individual que en tiempo de ejecución realiza las tareas de un programa.

SEI: Son las siglas en inglés de Software Engineering Institute (Instituto de Ingeniería de Software). Es un instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso de los Estados Unidos en 1984 para desarrollar modelos de evaluación y

mejorar en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los subsistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa de los Estados Unidos y administrado por la Universidad Carnegie Mellon.

Software: Término genérico utilizado en informática para designar programas o fragmentos de programas.

WSDL: Son las siglas de Web Services Description Language, formato que constituye un idioma descriptivo de los servicios web y un estándar del consorcio W3C (World Wide Web Consortium).

Anexos

Anexo # 1: Representación gráfica del conector evento, el gráfico (a) corresponde al caso en que hay un administrador de eventos, en tanto que el (b) corresponde a un bus de eventos.

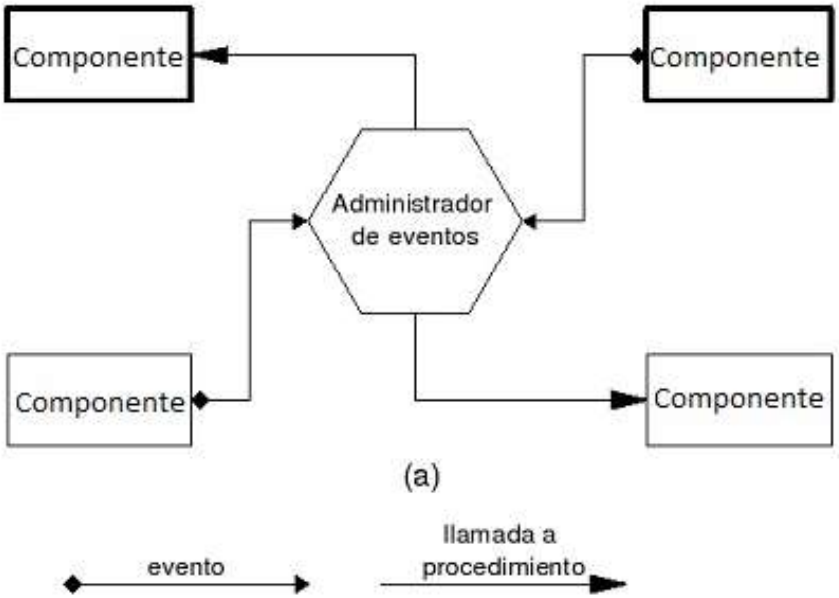


Figura 9 Conector evento en el caso de un administrador de eventos. Tomado de (Cristiá, 2006).

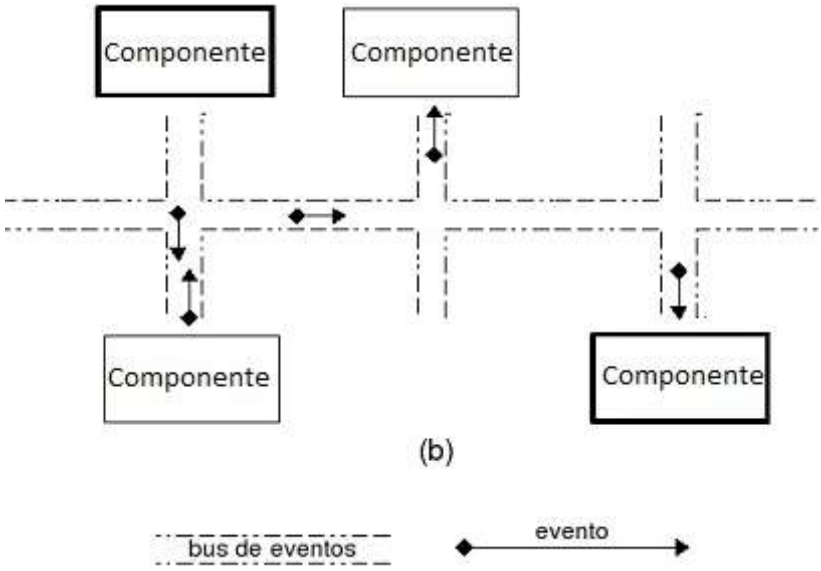


Figura 10 Conector evento en el caso de un bus de eventos. Tomado de (Cristiá, 2006).

Anexo # 2: Evaluación de cada indicador por cada especialista de acuerdo a la escala definida anteriormente.

Indicador	Evaluación
Reutilizable	8
Integrable	9

Tabla 2 Evaluación de los indicadores definidos por el especialista # 1.

Indicador	Evaluación
Reutilizable	9
Integrable	9

Tabla 3 Evaluación de los indicadores definidos por el especialista # 2.

Indicador	Evaluación
Reutilizable	9
Integrable	10

Tabla 4 Evaluación de los indicadores definidos por el especialista # 3.

Indicador	Evaluación
Reutilizable	10
Integrable	9

Tabla 5 Evaluación de los indicadores definidos por el especialista # 4.

Indicador	Evaluación
Reutilizable	8
Integrable	9

Tabla 6 Evaluación de los indicadores definidos por el especialista # 5.

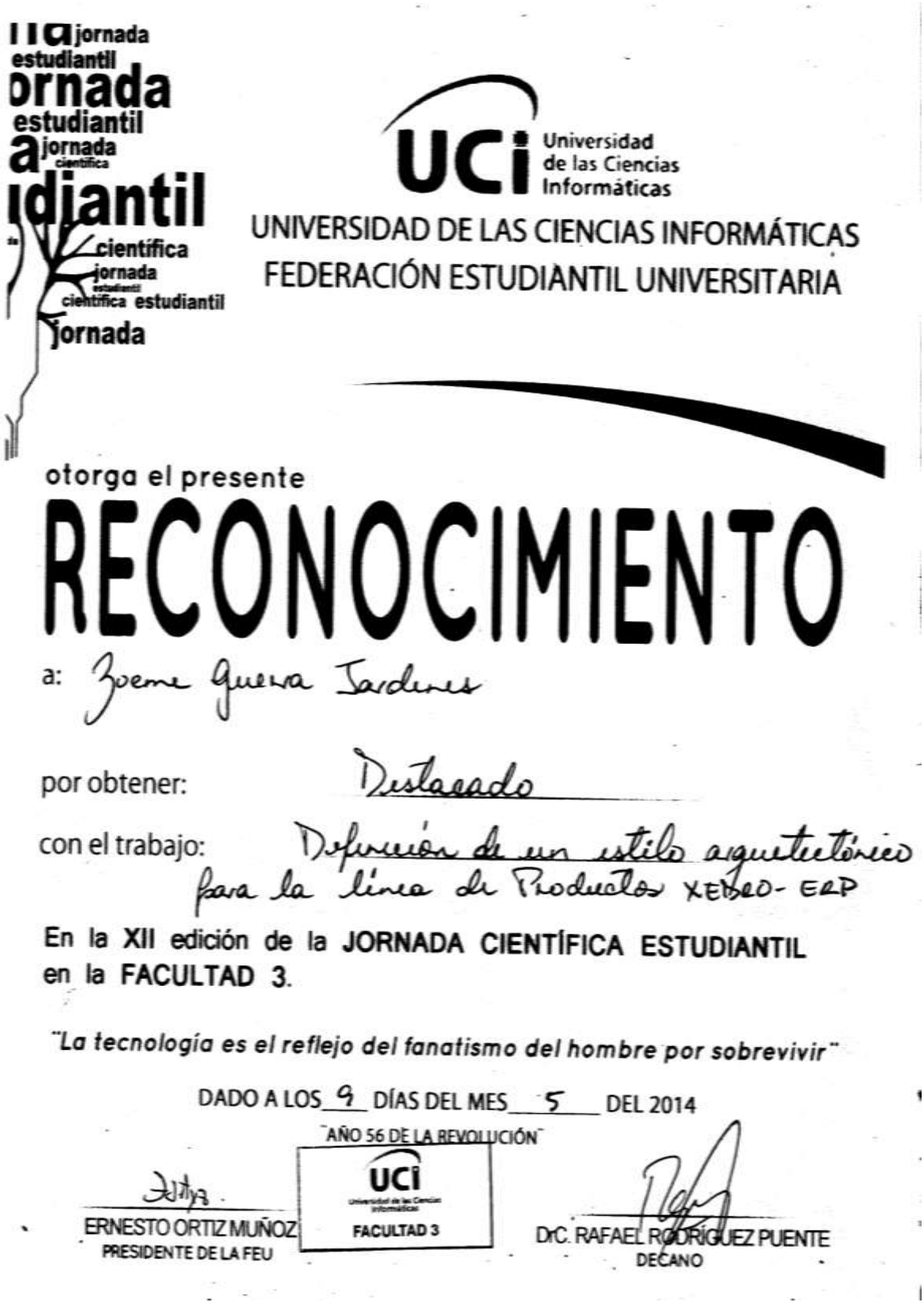
Indicador	Evaluación
Reutilizable	10
Integrable	9

Tabla 7 Evaluación de los indicadores definidos por el especialista # 6.

Indicador	Evaluación
Reutilizable	10
Integrable	10

Tabla 8 Evaluación de los indicadores definidos por el especialista # 7.

Anexo # 3: Premio de Destacado que se obtuvo con el presente trabajo en la XII Jornada Científica Estudiantil a nivel de Facultad.



XII jornada científica estudiantil

UCI Universidad de las Ciencias Informáticas

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FEDERACIÓN ESTUDIANTIL UNIVERSITARIA

otorga el presente

RECONOCIMIENTO

a: Zoeme Guerra Jardines

por obtener: Destacado

con el trabajo: Definición de un estilo arquitectónico para la línea de Productos XETEO- ERP

En la XII edición de la JORNADA CIENTÍFICA ESTUDIANTIL en la FACULTAD 3.

"La tecnología es el reflejo del fanatismo del hombre por sobrevivir"

DADO A LOS 9 DÍAS DEL MES 5 DEL 2014

"AÑO 56 DE LA REVOLUCIÓN"

ERNESTO ORTIZ MUÑOZ
PRESIDENTE DE LA FEU

UCI Universidad de las Ciencias Informáticas
FACULTAD 3

DrC. RAFAEL RODRÍGUEZ PUENTE
DECANO

Anexo # 4: Premio de Relevante que se obtuvo con el presente trabajo en la XII Jornada Científica Estudiantil a nivel de Universidad.

UCI

Universidad de las Ciencias Informáticas
Federación Estudiantil Universitaria

Otorgan el presente:

RECONOCIMIENTO

A: Definición de un estilo arquitectónico para la línea de productos XEDRO-ERP.

por obtener la condición de

RELEVANTE

en la XII Edición de la Jornada Científica Estudiantil.

"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica"

Aristóteles

Dado a los 22 días del mes de mayo de 2014.

Ailyn Febles Estrada
Vicerrectora de Investigación y Postgrado

Víctor Gabriel González Cardoso
Presidente de la FEU