Estilo arquitectónico para el sistema integrado de gestión Cedrux

Yusnier Matos Arias
ymarias@uci.cu
Universidad de las Ciencias Informáticas

Nemury Silega Martínez
nsilega@uci.cu
Universidad de las Ciencias Informáticas

RESUMEN

Cedrux, es un sistema integrado de gestión que se desarrolla en la Universidad de las Ciencias Informáticas. Estos sistemas poseen tres características que los identifican: integrales, modulares y adaptables. Durante su desarrollo se ha determinado la necesidad de diagnosticar dichas características a partir de análisis automatizados sobre su arquitectura en diferentes fases de su construcción. Para automatizar los análisis arquitectónicos es necesario antes: definir el estilo arquitectónico, formalizarlo de manera que pueda ser interpretado por un sistema informático, definir las métricas correspondientes e implementar una herramienta que permita aplicarlas. Este trabajo muestra la definición del estilo arquitectónico de Cedrux como primer paso en el desarrollo de la herramienta para el análisis arquitectónico del mismo. Se basa en la definición del vocabulario arquitectónico y las restricciones estructurales, sintácticas y semánticas de los elementos que conforman la arquitectura. Finalmente se exponen las ideas del trabajo futuro a partir de los resultados mostrados.

PALABRAS CLAVE: Análisis arquitectónico, componentes, conectores, estilo arquitectónico.

INTRODUCCIÓN

El sistema de planificación de recursos empresariales (ERP) Cedrux, también como sistema integrado de gestión (SIG) Cedrux, por definición, debe ser modular, integral y adaptable (Codorniú 2011). CEDRUX debe satisfacer un conjunto de requisitos no funcionales. Entre ellos: a) debe ser modular: parte del entendimiento de que una empresa es un conjunto de departamentos que se encuentran interrelacionados por la información que comparten y que genera a partir de sus procesos. (Codorniú 2011) b) debe ser integral: debe entenderse que todos los departamentos se relacionan entre sí y que el resultado de un proceso es punto de inicio del siguiente (Codorniú 2011), c) debe ser adaptable: las prácticas soportadas por estos sistemas se basan en supuestos generales acerca de cómo una empresa debe operar (Ramírez 2004), sin embargo, deben permitir la particularización hacia las características específicas de cada empresa.

El diseño de Cedrux debe soportar las necesidades particulares de este tipo de sistema. Las características mencionadas dependen en gran medida de la cohesión y el acoplamiento con que cuentan los elementos del sistema. Como principio, es importante lograr una alta cohesión y un bajo acoplamiento en aras de las cualidades descritas anteriormente.

Sin embargo, durante su desarrollo se ha dificultado la obtención de valores óptimos para estas cualidades, fundamentalmente para la adaptabilidad y otros indicadores asociados a este como: mantenibilidad, flexibilidad y escalabilidad. Tanto es así que durante los procesos de verticalización que han tenido lugar, se han presentado problemas a la hora de ejecutar variaciones estructurales sobre el sistema como parte de la propia adaptación que se requiere. En este contexto se puede entender como verticalización el conjunto de actividades, tanto de gestión como de implementación, que se realizan para adaptar funcionalidades generales del sistema hacia particularidades específicas de un cliente.

Los problemas más comunes se relacionan precisamente con el alto acoplamiento existente entre elementos arquitectónicos del sistema. De lo anterior se puede apreciar que estas variaciones recaen directamente sobre la arquitectura de software (en lo adelante AS o simplemente: arquitectura) que soporta al mismo, y donde desempeñan un rol fundamental los componentes de dicha AS.

La arquitectura de software, de acuerdo con (Bass, Clements y Kazman 2003), se define como sigue: "(…) la estructura o estructuras del sistema, constituida por los elementos de software, las propiedades externamente visibles de estos elementos y las relaciones entre ellos". Otra definición muy referenciada es la que se ofrece en IEEE (2000), en el std. 1471-2000, este define la AS como: "La organización fundamental de un sistema encarnada en sus componentes, sus relaciones entre sí y con el entorno, así como los principios que guían su diseño y evolución". Estas dos definiciones no están alejadas entre sí. En la primera, el término elemento puede ser interpretado como componente, siendo una definición más general de los bloques de construcción principales que estos representan en la definición del IEEE.

A partir de lo anterior se adopta en este trabajo la definición de AS del IEEE teniendo en cuenta su enfoque hacia componentes y el posterior uso de este término a lo largo de la investigación. Para ello se toma la definición de componente adoptada por el Software Engineering Institute (SEI): "Un componente es un fragmento de un sistema de software que puede ser ensamblado con otros fragmentos para formar piezas más grandes o aplicaciones completas" (Bachman et al. 2000). Los requisitos descritos con anterioridad refuerzan la orientación a componentes como una de las características fundamentales de la AS de Cedrux (Fernández 2011).

Como resultado de análisis realizados por varios integrantes del equipo de desarrollo del sistema Cedrux, de distintos roles (arquitectos, analistas, desarrolladores avanzados), se han identificado como causas de la problemática explicada las siguientes: a) resulta complejo, durante el diseño de la AS, tener en cuenta todos los elementos necesarios para lograr valores deseados de adaptabilidad, mantenibilidad, escalabilidad y flexibilidad, dada la cantidad de factores que intervienen así como las relaciones entre estos, además de la cantidad de componentes del sistema, b) en la práctica resulta inviable realizar más de una variante de diseño de la arquitectura para establecer comparaciones de factibilidad, por lo expresado en el punto anterior.

Para resolver la problemática anterior, se propone desarrollar una herramienta que a partir de modelos de la arquitectura, y haciendo uso de métricas para medir adaptabilidad, mantenibilidad, escalabilidad y flexibilidad, permita diagnosticar dichos factores cuantitativamente. Sin embargo, para el desarrollo de esta herramienta es necesario antes: establecer las métricas adecuadas, formalizar la arquitectura para que pueda ser analizada

por un sistema informático y por tanto, contar con la definición de los elementos que serán analizados.

El objetivo de este trabajo es definir los elementos que componen la AS del sistema Cedrux, las interacciones permitidas (válidas) de estos así como sus comportamientos respectivos. Los resultados de este trabajo constituyen el primer paso en pos del desarrollo de una herramienta para la realización de análisis arquitectónicos de Cedrux. Una vez definidos estos elementos, se podrá llevar a cabo la formalización (en un lenguaje formal o matemático) de los mismos. Este punto será tratado como parte del trabajo futuro en este documento.

ESTILO ARQUITECTÓNICO

Un estilo arquitectónico (EA) típicamente 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 (Garlan 2003).

Es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales (Billy 2004).

Según (Abowd, Allen y Garlan 1993) en un estilo es importante separar dos elementos fundamentales: a) dominio sintáctico abstracto de la descripción arquitectónica y b) dominio semántico de significados arquitectónicos.

En la definición anterior, la sintaxis abstracta no está asociada a una arquitectura en específico. En lugar de esto define a la AS en términos de tres clases abstractas bases: componentes, conectores y configuraciones (Abowd Allen y Garlan 1993). Por otro lado, el dominio semántico se centra en describir el significado de la sintaxis abstracta para un estilo específico, ejemplo Tuberías-Filtros (Abowd Allen y Garlan 1993).

Los estilos por lo general proveen cuatro argumentos (Monroe, Kompanek, Melton, Garlan 1997):

1. Un *vocabulario* de elementos de diseño: tipos de componentes y conectores, como pueden ser filtros, tuberías, clientes, servidores, etc.
2. Reglas de diseño o *restricciones* que determinan las composiciones permitidas de esos elementos.
3. Interpretación *semántica*, por lo que las composiciones de elementos de diseño, debidamente limitados por las reglas de diseño, tienen significados bien definidos.
4. *Análisis* que pueden ser realizados en los sistemas construidos con el estilo.

Una definición más general para estilo arquitectónico es la que se ofrece en Taylor, Medvidovic y Oreizy (2009), tomada originalmente de Taylor, Medvidovic y Dashofy (2009) : "(…) colecciones de decisiones de diseño que 1) son aplicables en un contexto de desarrollo dado, 2) restringen las decisiones de diseño arquitectónicas específicas para un sistema dentro de dicho contexto, y 3) persigue cualidades que benefician los sistemas resultantes".

Como se aprecia en la definición anterior, se puede describir el estilo arquitectónico (EA) como una colección de decisiones de diseño (DS). Sin embargo no todas las DS

caracterizan al EA, sino que son las DS principales las que lo hacen (Dashofy 2007). Estas DS son las que afectan horizontalmente al sistema, por lo tanto implican a la mayoría de los elementos que conforman el mismo (Dashofy 2007).

Al hablar de EA (y de AS en general) se considera una buena práctica referirse en un principio a los clásicos. Es por esta razón que se presenta un resumen de dos de los EA más referenciados actualmente como son: Tuberías y Filtros y Chiron-2 o C2.

A. Tuberías y Filtros
Los sistemas basados en el estilo Tuberías y Filtros pueden ser estructurados como una colección de transformaciones incrementales. Las subtransformaciones son combinadas de manera que la salida de una constituye la entrada de la otra. La aplicación consiste entonces en la combinación de todas las transformaciones. Este tipo de sistemas se conoce como flujo de dato de grano grueso o de tuberías y filtros (Allen y Garlan 1992).

La definición de este EA se realiza en tres pasos: 1) definición del elemento Filtro, que constituye el componente de la arquitectura, 2) definición de las tuberías, que son los conectores básicos y 3) especificación de cómo los filtros y las tuberías se combinan para formar un sistema completo. Para cada uno de estos elementos se caracterizan sus propiedades estáticas y dinámicas. Finalmente se muestra el comportamiento de cada elemento ante los cambios de estado. La descripción de este estilo se realiza utilizando el lenguaje de especificación Zeta (Z) (Allen y Garlan 1992). Detalles de este lenguaje se explican en Allen y Garlan (1992).

B.Chiron-2 (C2)
El estilo Chiron-2 está diseñado para soportar las necesidades particulares de aplicaciones que tienen un aspecto de interfaz gráfica de usuario, aunque tiene potencial para soportar otros tipos de aplicaciones (Taylor et al. 1996). Este estilo soporta un paradigma en el que los componentes de interfaz de usuario tales como diálogos, modelos gráficos estructurados de diferentes niveles de abstracción y gestores de restricciones, pueden ser reutilizados de mejor manera (Taylor et al. 1996).

Los indicadores de calidad fundamentales que persigue este estilo son heterogeneidad, concurrencia y composición (Taylor et al. 1996). Para esto propone comportamientos de sus componentes que hacen a estos independientes en gran medida de la tecnología en que se implementan.

La descripción del estilo se hace a partir de la definición de los componentes y conectores como elementos fundamentales del mismo, donde la composición de estos elementos representa la arquitectura. Se especifica además una serie de principios que pautan las combinaciones válidas de los componentes y conectores.

Los detalles de las definiciones de este estilo se presentan en Taylor et al. (1996).

DECISIONES DE DISEÑO

Kruchten también considera la descripción de la AS a partir de un conjunto de DS 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 EA que guía la organización de los anteriores (Kruchten 2004).

En la referencia Kruchten (2004) se propone además un modelo de DS donde estas son clasificadas. Se sugieren atributos relevantes para cada clasificación así como relaciones entre las DS.

Las DS se clasifican en: Existenciales, las cuales comprenden las de tipo Estructural y de Comportamiento; propiedades, que declaran rasgos generales o cualidades del sistema, pueden ser Reglas (positivamente declaradas) o Restricciones (negativamente declaradas); Ejecutivas, estas no están directamente relacionadas con los elementos arquitectónicos y sus cualidades, sino con el entorno de negocio (Kruchten 2004).

Entre los atributos que se proponen para las DS se encuentran: descripción, razonamiento, alcance, estado, autor, fecha y hora, historial, costo y riesgo. Las relaciones que pueden tener unas con otras son: restricción, prohíbe, habilita, subsume, conflicto con, sobrescribe, comprende, entre otras (Kruchten 2004).

ESTILO ARQUITECTÓNICO DE CEDRUX

En este capítulo se presenta el EA de Cedrux así como el razonamiento detrás de cada una de las decisiones de diseño que dan lugar a este. La descripción del mismo se realiza siguiendo las propuestas de Garlan (2003) al respecto pero teniendo en cuenta los puntos de vista de otras fuentes abordas con anterioridad en este documento. De manera general el estilo 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.

Los elementos claves que conforman el estilo son componentes y conectores. Además de un conjunto de reglas que pautan las configuraciones válidas de estos componentes y conectores. Cada una de estas configuraciones representa la descripción de una arquitectura, que en esencia son instancias del estilo definido.

A partir de lo anterior se definen tipos o categorías de componentes de acuerdo a la función que tienen estos en el sistema. 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. Cada uno de estos tipos de componentes y conectores define un comportamiento determinado para sus instancias. Más adelante, al exponer las restricciones del estilo, se mostrarán reglas de compatibilidad entre algunos de estos tipos de componentes y conectores.

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. Ejemplos de estos casos son las invocaciones de funciones del marco de trabajo desde componentes que no forman parte de este.

Por su parte los conectores pueden desempeñar varios roles dentro de las comunicaciones en las que intervienen. Pueden ser comunicadores, coordinadores o facilitadores. Los comunicadores se encargan de mediar la invocación de servicios entre componentes no agregados entre sí, validan la compatibilidad entre interfaces de comunicación al identificar parámetros incorrectos. Los coordinadores intervienen en la comunicación entre un componente agregado y uno contenido en este. Su única función en estos casos es conectar

ambas interfaces, no realiza validación de compatibilidad puesto que simplemente transfiere la responsabilidad hacia un componente conocido por él. Los conectores facilitadores implementan mecanismos para la interoperabilidad con componentes externos al sistema. Por lo general se enfocan en estándares o contratos más estrictos y hacen uso de patrones de diseño específicos: buzón de mensajes, recursos compartidos, etc.

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

Para la presentación del estilo, se aplica una interpretación de las definiciones de (Garlan 2003; Taylor, Medvidovic y Dashofy 2009). En estas definiciones se propone describir el estilo mediante un vocabulario de tipos para componentes, conectores y propiedades, además de reglas que rigen las configuraciones entre estos elementos. Estas reglas pueden verse como decisiones de diseño. A partir de esto, se define el vocabulario de componentes, conectores y propiedades para el estilo así como las restricciones de diseño correspondientes. La exposición de estas decisiones de diseño se hace siguiendo la ontología propuesta en Kruchten (2004) adaptada para este trabajo. Esta adaptación se centra en la selección de los atributos que se consideran necesarios.

Componentes:
Tecnológico: implementan funcionalidades puramente tecnológicas. Por lo general esta parte del sistema es responsabilidad de un framework o marco de trabajo que establece los mecanismos de comunicación, caché, transacciones, etc.

Negocio: Están directamente relacionados con funcionalidades de negocio dado que implementan los requerimientos del mismo

Dominio: implementan funcionalidades suplementarias para el negocio. Estos componentes pueden tener varias funciones. No obstante, no se dividen en otras categorías puesto que el comportamiento en el sistema es prácticamente el mismo. Su diferencia radica fundamentalmente en el tipo de información con que trabajan. Pero la forma de recibirla, brindarla, así como la manera en que la procesan es la misma. Su función fundamental es llevar a cabo cálculos o permitir configuraciones requeridas para realizar las funcionalidades de negocio. Estos cálculos y configuraciones tienen un comportamiento horizontal para el sistema, por lo que se encapsulan en componentes especializados.

Prueba: son componentes que tienen un tiempo de vida limitado. Funcionan como generadores de datos para probar funciones específicas dentro del sistema con el fin de simular salidas específicas o integraciones entre componentes durante las pruebas. No poseen puertos de entrada, solamente de salida.

Vertical: se crean para sustituir funcionalidades con el fin de adaptar los componentes actuales a negocios específicos evitando cambios en otros componentes. Abstraen las modificaciones estructurales durante los procesos de verticalización. Cuentan con puertos de entrada y de salida.

Conectores:
IoC: Pueden interpretarse como un tipo especial de componentes tecnológico cuyo fin es mediar la integración de componentes. Su funcionamiento se basa en la implementación del patrón Inversión de Control. Intervienen en comunicaciones inter-componentes no agregados entre sí.

Servicio Web: se usan para la comunicación con sistemas externos. Implementan el estándar WSDL. Intervienen en comunicaciones inter-componentes no agregados entre sí.

PC (ProcedureCall, del inglés): se usan para la comunicación entre dos funciones de un componente simple. Su ventaja fundamental es 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). Intervienen en comunicaciones intra-componente.

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

Restricciones
Las restricciones representan las reglas sintácticas y semánticas a las que responden cada una de las instancias del EA. 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.

Las restricciones de las que se trata en este trabajo responden a decisiones de diseño clasificadas en: Estructurales, de Comportamiento, Restricciones, y Guías. De acuerdo a las definiciones de Kruchten (2004). Es importante almacenar junto con cada decisión, el razonamiento que le dio origen, así como las ventajas o influencias sobre determinados atributos e indicadores de calidad. Además de estos atributos se incluirán las relaciones entre decisiones en caso que sean identificadas, los riesgos de su aplicación y un código que servirá para identificarlas más fácilmente en la práctica.

Las relaciones entre decisiones de diseño pueden ser: a) Restringe (Rg), b) Tiene conflicto con (Cf), c) Está relacionada con (Rn), d) Comprende (Cp). Detalles de estas relaciones pueden verse en Kruchten (2004).

A modo de ejemplo, se muestra a continuación un subconjunto de las decisiones de diseño devenidas en restricciones del estilo.

A00: **Los elementos mediante los cuales se debe describir la arquitectura son: componentes y conectores**.
Razonamiento: La arquitectura es descrita mediante configuraciones de componentes y conectores que responden a reglas sintácticas y semánticas que rigen la validez de dichas descripciones arquitectónicas.

Esto contribuye con la modularidad del sistema a la vez que separa la lógica computacional de los componentes y los mecanismos de comunicación de estos.
Riesgo: Hay comportamientos de la arquitectura cuya descripción formal se complejiza puesto que pueden ser necesarios otros elementos arquitectónicos que por lo general se tienen en cuenta en niveles de abstracción más bajos. Ejemplos de estos son patrones de diseño usados durante la implementación.
Categoría: Estructural, Restricción.
Relación:Rn (A01, A02).

A01: **Los componentes, por su función en el sistema, se clasifican en: Tecnológicos, Negocio, Dominio, Prueba, Verticales**.
Razonamiento: Cada una de estas categorías está explicada con anterioridad en el documento.
Categoría:Estructural, Restricción.
Relación:Rn (A00, A02).

A02: **Los tipos de conectores del estilo son: IoC, Servicios Web, Llamadas a Procedimientos (PC) y Bindings.**
Razonamiento: Cada uno de estos tipos de conectores determina un rol en un escenario de comunicación dado.
Comunicadores: IoC, PC y Servicios Web.
Coordinadores: *Bindings*.
Facilitadores: Servicios Web. Cuando la comunicación es con sistemas externos y utilizan mecanismos como buzón de mensajes o recursos compartidos. Por lo general implementan estándares como xml.
Categoría: Estructural, Restricción.
Relación: Rn (A00, A01)

A03: **Los componentes se comunican con el entorno a través de puertos de Entrada y de Salida.**
Razonamiento: Los componentes publican sus funcionalidades a través de interfaces con dos fines fundamentales: brindar servicios y consumir servicios.
Categoría: Estructural, Restricción.
Relación: Cp (A04)

A04: **Los puertos de Entrada y los de Salida son distintos en su comportamiento, deben ser diferenciados.**
Razonamiento: Es importante diferenciar los puertos de Entrada y de Salida puesto que su comportamiento y significados en el sistema son distintos. Los puertos de entrada representan funcionalidades que provee el componente y que pueden ser accedidas por otros componentes. Sin embargo, los puertos de Salida indican al entorno cuáles son las necesidades del componente para llevar a término alguna funcionalidad. Ambos tipos de interfaces deben quedar explícitos en la descripción de los componentes, en caso que las posea. Existen componentes que no poseen puertos de Entrada, esto debe ser descrito como otra restricción.
Categoría:Estructural, Restricción.
Relación:Rn (A03, A05)

A05: **Los componentes Tecnológicos no poseen puertos de Entrada, puesto que no consumen servicios.**
Razonamiento: La razón de ser de los componentes Tecnológicos es proveer la base tecnológica del sistema. Por lo general las funciones que proveen deben tener comportamientos (requisitos no funcionales) específicos. Dado que son muy usados y tienen un alcance horizontal en el sistema (afectan a todos los componentes del mismo) deben ser rápidos, precisos y atómicos, por lo que se deben evitar las dependencias entre estos en aras de disminuir las llamadas innecesarias.
Riesgo: Al tratar de evitar este tipo de comportamiento pueden convertirse en componentes demasiado grandes, con muchas funcionalidades relacionadas entre sí.
En trabajos futuros se definirán pautas más específicas para este comportamiento.
Categoría: Comportamiento, Guía.

A06: **Todos los componentes deben poseer al menos un puerto de Salida, mediante el cual ofrecen un servicio.**
Razonamiento: Un componente que no ofrece servicios tiene un sentido cuestionable en el sistema. Puede ser un indicador de un mal diseño. Se debe revisar la probabilidad de que sus funcionalidades formen parte de otro componente.
Categoría: Comportamiento, Guía.

A07: **Un puerto de Entrada solamente puede estar conectado en un momento dado con un único puerto externo.**
Razonamiento: Un puerto de Entrada puede estar conectado a un único puerto externo. Se refiere a un puerto de otro componente.
Puede estar conectado, no obstante, a varios puertos internos, siempre que estos sean de salida. A este tipo de conexión se le llama Transición. En realidad estos puertos no estarían conectados directamente, sino que existe de por medio una o varias funciones que responden a un puerto de Salida, sin embargo, esta función requiere datos de otros componentes para responder a la petición inicial y por tanto invoca un servicio del mismo. Esa invocación debe hacerla a través de un puerto de Entrada propio. Desde un nivel de abstracción alto se interpreta como una conexión entre ambos puertos: de Entrada y de Salida.
Categoría: Comportamiento, Restricción.

A08: **Los componentes de Dominio no deben consumir servicios de otros componentes de Dominio.**
Razonamiento: En aras de un menor acoplamiento. Deben ser componentes atómicos. Las dependencias con otros componentes de este tipo solo pueden ser jerárquicas.
Riesgo:Revisar A05.
Categoría:Comportamiento, Guía.

A09: **Los conectores median la comunicación entre componentes a través de roles.**
Razonamiento: Se recomienda que en una conexión esté bien definido qué rol desempeña cada parte. Por lo general los roles de los conectores están implícitos y no se tienen en cuenta en las descripciones arquitectónicas.
Categoría:Comportamiento, Guía.

A10: **Un componente de Negocio no debe tener subcomponentes Tecnológicos en ningún nivel jerárquico.**
Razonamiento: Esto crearía dependencias por lo general innecesarias entre la parte computacional de los componentes y los mecanismos tecnológicos que debería proveer el marco de trabajo. Uno de los fines que se persigue es desacoplar las funcionalidades esenciales de los componentes de la base tecnológica. Esto permite mayor flexibilidad a la hora de realizar modificaciones o adaptaciones tecnológicas sin implicaciones en los componentes de Negocio y viceversa.
Categoría: Estructural, Restricción.

A11: **Los componentes tecnológicos deben ser simples.**
Razonamiento: La razón de ser de los componentes Tecnológicos es proveer la base tecnológica del sistema. Por lo general las funciones que proveen deben tener comportamientos (requisitos no funcionales) específicos. Dado que son muy usados y tienen un alcance horizontal en el sistema (afectan a todos los componentes del mismo) deben ser rápidos, precisos y atómicos, por lo que se deben evitar las dependencias entre estos en aras de disminuir las llamadas innecesarias.
Categoría: Estructural, Guía.

A12: **No se puede acceder directamente a los servicios que publica un componente que es parte de un componente contenedor.**
Razonamiento: Los componentes que pueden presentar estructuras jerárquicas no pueden ser accedidos directamente. Este acceso debe ser a través del componente que lo contiene, sin violar en ningún momento dicha estructura jerárquica.
Es recomendable permitir el acceso directamente solo si el componente que consume el servicio está en el mismo nivel jerárquico que el que lo brinda, y ambos comparten el mismo componente contenedor (el mismo padre).
Contribuye con la mantenibilidad y flexibilidad del sistema.
Riesgo: Implementar este mecanismo en la práctica puede resultar engorroso y contraproducente en términos de rendimiento, puesto que se debe tener registrada la relación jerárquica de todos los componentes, o proveer un mecanismo que la identifique dinámicamente. Ambos métodos pueden atentar contra los tiempos de respuestas del sistema, dado que demoran innecesariamente las llamadas a servicios.
Categoría: Comportamiento, Restricción.

A13: **Un componente que es parte de un componente contendor no puede acceder directamente a los servicios que publica otro componente no contenido en dicho componente contenedor.**
Razonamiento: El acceso debe realizarse mediante el componente que contiene al primero.
Categoría: Comportamiento, Restricción.

A14: **No deben existir ciclos entre componentes cuando estos sean traceables desde una petición inicial.**
Razonamiento: Siguiendo las transiciones o conexiones internas de un componente, a partir de una petición inicial desde un componente externo. Para dar respuesta a una petición, el componente que es accedido desde otro no debería invocar funcionalidades del que lo invocó anteriormente o de los que invoquen al anterior.
Categoría: Comportamiento, Restricción.
Relación: Rn (A15)

A15: **Se deben evitar ciclos entre componentes.**
Razonamiento: Los componentes de software, por lo general, son interdependientes. Ofrecen funcionalidades hacia otros componentes a la vez que utilizan las que estos ofrecen. Por tanto pueden existir ciclos entre varios componentes que necesitan unos de otros sin que dichas dependencias tengan que ver directamente unas con otras. Prohibir la existencia de este tipo de dependencias cíclicas en una arquitectura es en la práctica insostenible. Sin embargo la existencia de estas atenta contra valores óptimos de cohesión y acoplamiento, por tanto deben tomarse medidas para evitarlas.
Categoría: Estructural, Guía.
Relación: Cp(A14).

FUTURAS LÍNEAS DE TRABAJO

Para la validación del estilo se han realizado talleres en los que han participado arquitectos del sistema, así como analistas y desarrolladores avanzados. Estos talleres se han realizado con el fin de validar la correspondencia de la propuesta con las definiciones actuales sobre la AS de Cedrux y fundamentalmente con las brechas existentes en esta. Por tanto, en la propuesta se tienen en cuenta los aspectos actuales pero que se consideran correctos, y se agregan otros (estructurales y de comportamiento) que contribuyen con valores adecuados de las cualidades deseadas de la AS, además de permitir la formalización del estilo. Lo anterior constituye solo una parte de esta validación, pues se planea aplicar la propuesta en módulos relativamente pequeños, aunque no triviales, para analizar los resultados.

El próximo paso para el desarrollo de una herramienta informática que permita realizar los análisis arquitectónicos con mayor precisión y agilidad, y así diagnosticar las cualidades deseadas de la AS en distintas fases de su construcción, es formalizar el estilo propuesto matemáticamente. La construcción de la herramienta no debe llevarse a cabo "desde cero". Sino que se pretende utilizar las facilidades que brindan los Lenguajes de Descripción de Arquitecturas (ADL, Architecture Description Languages) existentes. Algunos de estos ofrecen funcionalidades para diagnosticar cualidades de arquitecturas específicas (Dashofy 2007).

El ADL propuesto para desarrollar la herramienta de análisis arquitectónico de Cedrux es ACME. Algunas de las razones que soportan esta propuesta son las siguientes:

1. Es uno de los más usados actualmente.
2. Cuenta con una herramienta gráfica multiplataforma basada en el IDE Eclipse (AcmeStudio) que es actualizada periódicamente por el equipo de desarrollo.
3. Provee una biblioteca (AcmeLib) que permite extender sus funcionalidades para incluir análisis arquitectónicos.
4. Permite la definición de estilos arquitectónicos sobre la base de elementos como componentes, conectores, configuraciones, y otros.
5. Provee un lenguaje basado en Lógica de Predicados de Primer Orden para formalizar la arquitectura, el cual permite, entre otros aspectos, especificar restricciones sintácticas y semánticas (mediante propiedades).

De forma simultánea a la formalización del estilo, se irán definiendo las métricas correspondientes. Una vez definidas, serán incluidas en la herramienta como extensiones de AcmeStudio. Finalmente, se validará la factibilidad de la herramienta desarrollada a través de su aplicación en el proceso de construcción de la arquitectura de Cedrux en las distintas etapas de este.

CONCLUSIONES

El EA propuesto no desecha las definiciones anteriores de la arquitectura de Cedrux. A pesar de que considerar que algunas de ellas influyen en valores no deseados de las cualidades abordadas en este documento. Se tuvo esto en cuenta para mantener la compatibilidad con los componentes desarrollados hasta el momento. Sin embargo se deben ir refinando paulatinamente.

La definición de los elementos fundamentales que componen la arquitectura de Cedrux, contribuye con el proceso de obtención de componentes, que es crucial en la construcción de la arquitectura. Provee una guía para la identificación correcta de los componentes arquitectónicos en dependencia de las funcionalidades que implementan y puede constituir un primer paso en la formalización de este proceso.
Este trabajo pudiera ser extendido a otros proyectos de desarrollo de software haciendo las variaciones correspondientes, o al menos servir como guía para ello.

Al formar parte de un trabajo más amplio, establece las definiciones necesarias para el próximo paso en este último, que es el desarrollo de una herramienta para el análisis arquitectónico de Cedrux. En un principio enfocado a: adaptabilidad, mantenibilidad, escalabilidad y flexibilidad, pero que puede extenderse para diagnosticar otras cualidades de la arquitectura.

REFERENCIAS

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

Allen R. y Garlan, D. (1992) *Towards Formalized Software Architectures*. Carnegie Mellon University, Pittsburgh, PA.

Bachman, F., Bass, L., Buhman, C., Cornella-Dorda, S., Long, F., Robert, J., Seacord, R. y Wallnau, K. (2000) Volume II: Technical Concepts of Component-Based Software Engineering, Software Engineering Institute, Technical Report CMU/SEI-2000-TR-008.

Bass, L., Clements, P. y Kazman, R. (2003) *Software Architecture in Practice*, Segunda Edición ed: Addison Wesley.

Billy Reynoso, C. (2004) *Introducción a la Arquitectura de Software*, Buenos Aires.

Codorniú, C. L. (2011) "Solución arquitectónica de la Configuración General del sistema para la parametrización de negocio de Cedrux " en centro de informatización de la gestión de entidades. Ciudad de la Habana, Cuba: Universidad de las ciencias informáticas.

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

Fernández, O.L. (2011) "Propuesta metodológica para la obtención de los componentes de software en los proyectos del sistema Cedrux". Informática Aplicada en Centro de Informatización de la Gestión de Entidades. Universidad de las Ciencias Informáticas, La Habana.

Garlan, D. (2003) "Formal Modeling and Analysis of Software Architecture: Components, Connectors, and Events," Formal Methods for Software Architectures, Lecture Notes in Computer Science Volume 2804, pp 1-24.

IEEE (2000) *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE-Std-1471-2000.

Kruchten, P. (2004) "An Ontology of Architectural Design Decisions in Software-Intensive Systems" en Jan Bosch (ed.), Proc. of the 2nd Workshop on Software Variability Management, Groningen, NL, Dec. 3-4

Monroe, R. T., Kompanek, A., Melton, R. y Garlan, D. B. (1997) "Architectural Styles, Design Patterns, and Objects," IEEE Software, pp. 43-52.

Ramírez Correa, P. (2004) "Rol y Contribución de los Sistemas de Planificación de los Recursos de la Empresa (ERP)," Tesis doctoral. Sevilla.

Taylor, R. N. y Medvidovic, N., Oreizy, P. (2009) "Architectural Styles for Runtime Software Adaptation". Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on.

Taylor, R., Medvidovic, N. y Dashofy, E. (2009) *Software architecture: Foundations, theory and practice*, John Wiley & Sons Ltd.

Taylor, R. N., Medvidovic, N., Anderson, K. M., Whitehead Jr. E. J., Robbins, J. E., Nies, K. A., Oreizy, P. y Dubrow, D. L. (1996) "A Component- and Message-Based Architectural Style for GUI Software" in IEEE Transactions on Software Engineering, vol. 22, no. 6, 390-406 p.