Revista Cubana de Ciencias Informáticas
Vol. 13, No. 1, Enero-Marzo, 2019
ISSN: 2227-1899 | RNPS: 2301
Pág. 143-157
http://rcci.uci.cu

Tipo de artículo: Artículo de revisión
Temática: Ingeniería y gestión de software
Recibido: 14/05/2018 | Aceptado: 26/10/2018

Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software

Review of conceptual elements at representation of software reference architectures

Miguel Angel Sánchez Palmero [0000-0002-9870-5193]*****, **Nemury Silega Martínez** [0000-0002-8436-5650], **Olga Yarisbel Rojas Grass** [0000-0002-0642-6673]

CEIGE, Facultad 3, Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, Km 2 ½ La Lisa, La Habana, Cuba. {masanchez, nsilega, yarisbel}@uci.cu

***Autor para correspondencia**: masanchez@uci.cu

Resumen

La representación de conocimiento de las arquitecturas de referencias de software abarca elementos que particularizan una arquitectura de software, así como la descomposición y definición estándar de sus componentes. La presente investigación identificó las formas comunes para representar las arquitecturas de referencias de software; aportó, además, los elementos relevantes de las arquitecturas y sus insuficiencias. Como resultado de la revisión se obtuvo que existen diferentes formas de representación para las arquitecturas de referencias de software, descritas usualmente en lenguaje natural, limitando el uso de herramientas o notaciones utilizadas para el análisis y reutilización de la información, así como el chequeo de la correctitud de las especificaciones. Los hallazgos relacionados con las arquitecturas de referencias de software se tuvieron en cuenta para el desarrollo de una propuesta que permitió representar el conocimiento arquitectónico relativo a las decisiones arquitectónicas y su razonamiento.

Palabras clave: arquitectura de referencia de software, concepto, representación de conocimiento.

Abstract

The knowledge representation of the software reference architectures covers elements that particularize software architecture, as well as the decomposition and standard definition of its components. The present research identifies the common ways to represent software reference architectures; it also contributes the relevant elements of the architectures and their insufficiencies. As a result of the review it was obtained that there are different forms of

Revista Cubana de Ciencias Informáticas
Vol. 13, No. 1, Enero-Marzo, 2019
ISSN: 2227-1899 | RNPS: 2301
Pág. 143-157
http://rcci.uci.cu

representation for software reference architectures, usually described in natural language, limiting the use of tools or notations used for the analysis and reuse of information as well as the verification of correctness of the specifications. The findings related to the architecture of software references will be taken into account for the development of a proposal that allows representing the architectural knowledge related to architectural decisions and their reasoning.
Keywords: *concept, knowledge representation, references architectures.*

Introducción

La Ingeniería de Software ha permitido que junto a la disciplina de implementación se desarrolle el modelado de negocio, análisis y diseño, pruebas, entre otras disciplinas que por etapas dejan sus huellas en el proceso de desarrollo de software. En la incorporación y consolidación de estos elementos al proceso de desarrollo de software también se añaden los elementos arquitectónicos, abarcando la estructura de alto nivel de la organización del sistema, conocida como la arquitectura del software

Sobre la base de la bibliografía consultada existe un gran número de definiciones, entre las más empleadas se destaca la propuesta por el Instituto de Ingeniería de Software, donde se refiere a la Arquitectura de Software como “las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos" (Bass, Clements, & Kazman, 2003). En otro de los conceptos, la Arquitectura de Software es definida como un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores, analistas y todo el conjunto de desarrolladores del software compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software (Carpio Encalada & Cango, 2016).

El IEEE *Working Group on Architecture* la define como el concepto de más alto nivel de un sistema en su entorno. También incluye el "ajuste" con la integridad del sistema, con las restricciones económicas, con las preocupaciones estéticas y con el estilo. No se limita a un enfoque interior, sino que tiene en cuenta el sistema en su totalidad dentro del entorno de usuario y el entorno de desarrollo, un enfoque exterior (Zarvić & Wieringa, 2014). En otro de los trabajos se define como un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema" (España & Fernando, 2016).

Independientemente de las discrepancias analizadas entre los diferentes investigadores sobre una definición de arquitectura, todos coinciden en que la arquitectura implica un alto nivel de abstracción. Esta definición implica que la arquitectura de un sistema define componentes y la interacción existente entre ellos, pero no detalles internos a esos componentes, que se puede considerar que no pertenecen a la arquitectura de la aplicación. La arquitectura del sistema se puede presentar desde un número no determinado de perspectivas, todas ellas válidas siempre que permitan analizar, comunicar o comprender el producto a desarrollar (Menéndez, 2015).

Una revisión de las definiciones esbozadas y su conocimiento aplicado en entidades desarrolladoras de software para dominios es lo que garantizaría elevados niveles de reutilización. La práctica de introducir la reutilización de los elementos arquitectónicos desde etapas tempranas del desarrollo, muestra el trabajo con las arquitecturas de referencias de software

Una arquitectura de referencia define la infraestructura común a todos los sistemas de un dominio en particular, los componentes o subsistemas que incluyen y las interfaces que deben ofrecer dichos componentes o subsistemas. Disponer de tal arquitectura facilita enormemente el desarrollo de nuevas aplicaciones dentro del dominio de solución en la cual se enmarca, pues permite por un lado la reutilización de modelos y componentes, y por otro ofrece un marco para el desarrollo de los mismos (Menéndez, 2015).

Las ARS surgen en un momento en que la Arquitectura de Software ha evolucionado a través de diferentes corrientes o escuelas. Por la composición, los elementos teóricos que se abarca y finalidad de los trabajos referenciados, según (Ochoa, 2011) se pueden distinguir a grandes rasgos en:

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

La escuela de arquitectura estructural constituye la corriente fundacional y clásica de la disciplina, con una marcada conceptualización de las estructuras y marcos analíticos de la arquitectura, convirtiéndola en punto de referencia. Por

Revista Cubana de Ciencias Informáticas
Vol. 13, No. 1, Enero-Marzo, 2019
ISSN: 2227-1899 | RNPS: 2301
Pág. 143-157
http://rcci.uci.cu

otro lado, la evolución de la AS hacia los modelos de vistas arquitectónicas, que de igual forma hereda las ARS, hace que varios modelos de ARS estén basados sobre este elemento arquitectural.

Materiales y métodos o Metodología computacional

El mapeo sistemático según (Martínez & Pino, 2016), define un proceso y una estructura de informe que permite categorizar los resultados que han sido publicados hasta el momento en un área determinada. Para esta investigación se determinó el uso de este método pues se considera menos exhaustivo, permitiendo tener una visión científica e identificar tendencias en la evolución de las arquitecturas de referencias de software. En el mapeo sistemático se definen preguntas de investigación, a partir de estas se obtienen los estudios primarios, se definen los criterios de análisis y finalmente se discuten los resultados

Para el mapeo se definieron las siguientes preguntas de investigación:

PI1: ¿Cuáles son los elementos relevantes que pueden tenerse en cuenta para la conceptualización de las arquitecturas de referencias de software?

PI2: ¿Cuál es el método más utilizado para formalizar las arquitecturas de referencias de software?

Las principales fuentes para esta investigación se basan en la IEEE y *Google Scholar*, que según el estudio de (Petersen, Vakkalanka, & Kuzniarz, 2015), la IEEE es una de las bases de datos referenciadas con mayor impacto en los resultados investigativos. *Google Scholar* garantiza el acceso a otros documentos que no se encuentran en las bases de datos organizadas pero que sí son de interés para la comunidad científica. Los trabajos seleccionados están relacionados con los lenguajes de descripción arquitectónica y las arquitecturas de referencias de software.

En el presente trabajo se investiga sobre la forma más utilizada actualmente para representar las arquitecturas de software y los elementos que pueden utilizarse para conceptualizar a las arquitecturas de referencias de software, con la visión de proponer el desarrollo de una ontología. Se presenta una breve revisión de los trabajos relacionados, se describe la metodología utilizada presentando los resultados y conclusiones.

Resultados y discusión

PI1: ¿Cuáles son los elementos relevantes que pueden tenerse en cuenta para la conceptualización de las arquitecturas de referencias de software?

De las diferentes corrientes o escuelas por las que ha atravesado el desarrollo de la Arquitectura de Software, es la Arquitectura estructural la que aporta una potente conceptualización de las estructuras, por lo que en los lenguajes de descripción arquitectónica (Reynoso) se muestran los elementos estructurales que en cada uno de ellos se desarrollan. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento (Reynoso, 2004).

UniCon (*Language for Universal Connector Support*) es un lenguaje de descripción arquitectónico (ADL) que permite a los diseñadores identificar los conectores y componentes. UniCon admite la evolución del diseño del sistema para implementar aplicaciones de la vida real (Abbasi, Butt, & Anjum, 2016). Establece un modelo para especificar la arquitectura del software haciendo uso de abstracciones predefinidas para describir los componentes, las interacciones entre dichos componentes (caracterizadas en este lenguaje como conectores) y los patrones que dirigen la composición de unos con otros para formar sistemas. Existe una gran variabilidad entre los distintos elementos que maneja el lenguaje, y entre los atributos específicos de cada uno de ellos y de los actores o roles que representan su interfaz (Velasco, Linero, de Lenguajes, & Sánchez, 2000). UniCon limita los protocolos para que sean tipos definidos, por ejemplo, acceso a datos, llamada de procedimiento, canalización, evitando así que los diseñadores especifiquen libremente sus tipos (Ozkaya & Kloukinas, 2013). El control estricto sobre los tipos componentes y conectores que pueden utilizarse y cuáles son las posibilidades de combinarlos es precisamente lo que permite a UniCon la generación de un sistema ejecutable a partir de la especificación arquitectónica. Este lenguaje no cuenta con una base formal para la validación de propiedades de las especificaciones. Además, carece de elementos para la reutilización de componentes o arquitectura, no se adapta a cambios en los requisitos y no permite la descripción de sistemas con una estructura que evolucione durante su ejecución (Velasco et al., 2000).

El lenguaje de descripción de la arquitectura *Wright* es conocido por su comportamiento preciso y formal de los conectores en el diseño de la arquitectura (Abbasi et al., 2016). Este lenguaje distingue también entre componentes y conectores y dispone de estructuras sintácticas diferenciadas para describir la interfaz de unos y otros. La interfaz de un componente está representada por medio de una serie de puertos (equivalentes por tanto a los actores de UniCon), la de un conector se describe también en este lenguaje por medio de una serie de roles. El principio que inspira el diseño de *Wright* no es la obtención de forma automática de un sistema ejecutable, sino que el objetivo consiste aquí en el análisis formal de los sistemas descritos. *Wright* ayuda a la especificación arquitectónica pero no apoya el resto de tareas propias del proceso de desarrollo (Velasco et al., 2000). *Wright* proporciona la especificación de arquitectura confiable y completa y proporciona un alto nivel de calidad de lenguaje (Clements, 1996). Este lenguaje no contempla

la simulación, ni la generación de código a partir de la especificación, no incorpora mecanismos de reutilización de componentes y conectores, ni de adaptación al cambio. Tampoco contempla la existencia de sistemas que presenten una arquitectura dinámica, que pueda evolucionar en tiempo de desarrollo (Velasco et al., 2000).

Darwin es un ADL que merece especial atención. En Darwin es posible describir arquitecturas dinámicas, es decir, que evolucionan durante la ejecución del mismo. Este lenguaje captura la mayor parte de la estructura del sistema en evolución mientras se mantiene su forma puramente declarativa, apoyado por las estructuras dinámicas (Hirsch, Kramer, Magee, & Uchitel, 2006). Los dos conceptos básicos que maneja el lenguaje son el de componente y el de servicio. Darwin proporciona dos mecanismos estrechamente relacionados para la descripción de arquitecturas dinámicas. El primero de ellos es la instanciación dinámica, que permite especificar la creación de componentes durante la ejecución del sistema. El segundo de estos mecanismos es la instanciación perezosa, por medio de la cual el componente que proporciona un servicio no se instancia hasta que un usuario intenta acceder a dicho servicio. La combinación de la instanciación perezosa con la composición recursiva permite la descripción de estructuras de tamaño variable. El lenguaje contempla la generación de código a partir de las especificaciones; sin embargo, no se dispone en el lenguaje de mecanismos que faciliten el refinamiento o la reutilización de las especificaciones (Velasco et al., 2000).

Se puede caracterizar a *Rapide* como un lenguaje de descripción de sistemas de propósito general que permite modelar interfaces de componentes y su conducta observable. Sería tanto un ADL como un lenguaje de simulación (Sepúlveda, 2017). En los prototipos que se generan se pretende que aparezcan reflejadas de forma explícita las propiedades de concurrencia, sincronización, flujo de información y temporización de la arquitectura que representan. Al igual que ocurre en Darwin, *Rapide* no incorpora el concepto de conector como parte del lenguaje, sino que modela los sistemas de software como un conjunto de componentes conectados mediante paso de mensajes o eventos. *Rapide* permite la descripción de arquitecturas dinámicas, aunque de forma restringida, el dinamismo en este lenguaje se limita a la descripción de sistemas que pueden presentar varias configuraciones alternativas. Por otra parte, el lenguaje incorpora un mecanismo de herencia o subtipado para la reutilización de especificaciones y su adaptación a cambios de los requisitos (Velasco et al., 2000).

C2 no es estrictamente un ADL sino un estilo de arquitectura de software que se ha impuesto como estándar en el modelado de sistemas que requieren intensivamente pasaje de mensajes y que suelen poseer una interfaz gráfica dominante (Sepúlveda, 2017). Al igual que en el resto de las propuestas, el concepto fundamental subyacente a C2 es que los sistemas de software están formados por componentes que se coordinan y comunican mediante paso de

mensajes (en concreto, peticiones de servicios y notificaciones de eventos). Sin embargo, la disposición de los componentes en C2 no es libre, sino que estos están organizados en capas. Uno de los aspectos más significativos de este lenguaje es la flexibilidad que presenta a la hora de realizar comprobaciones de tipos en las conexiones entre componentes. C2 contempla la generación de código a partir de la especificación, pero no así la verificación de propiedades. Así mismo, existe la posibilidad de describir arquitecturas dinámicas, contemplándose la inserción y eliminación de componentes en la arquitectura, pero está restringido por el estilo de arquitectura en capas que impone el lenguaje (Velasco et al., 2000).

Sobre este tema se incluye, además, el trabajo presentado por Ochoa en el que define los componentes estructurales de la disciplina de la Arquitectura de Software. Plantea que la disciplina de la Arquitectura de Software presenta cuatro elementos que concentran su fundamentación teórica estructural, Componentes, Conectores, Configuraciones y Restricciones, además de formalizar en siete grandes grupo tipológicos y no más de veinte ejemplares todas las problemáticas arquitectónicas existentes hasta la fecha en la industria actual del software, estos son denominados Estilos Arquitectónicos y facilitan la comunicación entre los especialistas y ayuda a organizar el proceso de abstracción arquitectónica.

En el trabajo de Sepúlveda se describen un conjunto de elementos que según su autor son los que deben modelar los ADL. Estos elementos arquitectónicos son: Componentes, Tipo de Componentes, Tipo de conectores, Puerto, Configuraciones o sistemas, Propiedades, Restricciones, Estilos, Propiedades No Funcionales y Evolución. Cada elemento tiene su rol bien especificado dentro del ADL y por ende en la arquitectura; existiendo, además, coincidencia con todos los elementos propuestos por Ochoa, se confirman como esenciales los conceptos Componente y Conector.

Tras la revisión de los trabajos relacionados con los ADL es posible plantear que en la conceptualización de las Arquitecturas de Software y Arquitecturas de Referencias de Software se deben tener en cuenta los 8 conceptos identificados: componente, conector, configuración, restricción, estilo arquitectónico, actor, rol y servicio. En los resultados obtenidos se indica que el elemento arquitectural base se considera al Componente, citado en el 100% de los trabajos. El concepto Conector está presente en el 50% de los trabajos, los conceptos Actor y Rol se manifiestan en el 33% de los trabajos y el resto de los elementos (Configuración, Restricción, Estilo Arquitectónico, Servicio) alcanzan el 17%. Por orden de relevancia e importancia en cuanto a elementos de conceptualización, a continuación, se detallan los dos conceptos más importantes según los trabajos citados: Componente y Conector.

Componente

En el desarrollo de software se entiende que un componente es un elemento de software pre-existente que implementa alguna funcionalidad, la cual puede ser accedida a través de interfaces bien definidas (Hernández, Velasco-Elizondo, & Benítez-Guerrero, 2016). Representa elementos computacionales primarios de un sistema. Intuitivamente, corresponden a las cajas de las descripciones de caja-y-línea de las arquitecturas de software. Ejemplos típicos serían clientes, servidores, filtros, objetos, pizarras y bases de datos. En la mayoría de los ADL los componentes pueden exponer varias interfaces, las cuales definen puntos de interacción entre un componente y su entorno (Sepúlveda, 2017). El *Software Engineering Institute* (SEI) de la Universidad *Carnegie Mellon* formuló una definición con el propósito de consolidar las diferentes opiniones acerca de lo que debía ser un componente de software: "un componente es una implementación opaca de funcionalidad, sujeta a composición por terceros y que cumple con un modelo de componentes" (Bachmann et al., 2000). Según Szyperski, un componente de software es una unidad de composición con interfaces especificadas contractualmente y dependencias de contexto explicitas únicamente. Además, puede ser desplegado de forma independiente y este sujeto a la composición por terceras partes y no tiene un estado observable (Quecan Rueda & Pulido Vásquez). Pressman define que los componentes forman la arquitectura del software y, en consecuencia, juegan un papel en el logro de los objetivos y de los requerimientos del sistema que se va a construir (Pressman, 2010). Como los componentes se encuentran en la arquitectura del software, deben comunicarse y colaborar con otros componentes y con entidades (otros sistemas, dispositivos, personas, etc.) que existen fuera de las fronteras del software. La aparición del término dentro de la arquitectura ha tenido un impacto positivo, permitiendo profundizar en la reutilización de software, además de propiciar el desarrollo de nuevos modelos y paradigmas de desarrollo orientados a componentes.

Conector

Representan interacciones entre componentes. Corresponden a las líneas de las descripciones de caja-y-línea. Ejemplos típicos podrían ser tuberías (pipes), llamadas a procedimientos, *broadcast* de eventos, protocolos cliente-servidor, o conexiones entre una aplicación y un servidor de base de datos. Los conectores también tienen una especie de interfaz que define los roles entre los componentes participantes en la interacción (Sepúlveda, 2017). Los conectores representan interfaces que proveen la capacidad de "usar" o "pasar datos a" un componente (Sommerville, 2011). El conector constituye un elemento con características especiales, el mismo puede ser, un método, un *buffer*, un fichero, una tabla en base de datos, un objeto en memoria, un componente de software etc. Cualquier elemento que permita intercambiar eventos o datos entre las partes de un software.

PI2: ¿Cuál es el método más utilizado para formalizar las arquitecturas de referencias de software?

El marco de referencia para la arquitectura empresarial de John Zachman provee 36 categorías necesarias para describir de manera completa cualquier cosa, especialmente, cosas muy complicadas. Abarca seis vistas especificadas o fases de abstracción desde seis perspectivas diferentes. De esta forma, distintas personas pueden ver lo mismo de manera diferente, esto crea una vista holística del entorno, siendo esta una capacidad sumamente importante (Ledesma Alvear, 2017).

El Modelo de Referencia para Procesamiento Distribuido Abierto (RM-ODP) es un estándar para la especificación arquitectónica de sistemas distribuidos. Este marco proporciona dos niveles de abstracción. El primer nivel define cinco puntos de vista: empresarial, información, computacional, ingeniería y tecnología. EL segundo nivel de abstracción define los términos que especifican la coherencia entre los puntos de vista (Balouki, Sbihi, & Balouki, 2015). RM-ODP es un marco de referencia neutral desde el punto de vista tecnológico, permite la coexistencia de diferentes soluciones que necesitan interoperabilidad y facilita la naturaleza evolutiva de la interoperabilidad en el tiempo (Milosevic & Bond, 2016).

C4ISR es el marco de referencia arquitectónico promovido por el Departamento de Defensa de Estados Unidos de América. En él se explica que las diferentes arquitecturas de sistemas de información militar deben integrarse en diferentes niveles, por ejemplo, el nivel de arquitectura física, el nivel de arquitectura de la aplicación y el nivel de arquitectura empresarial. En la actualidad, la integración de síntesis del sistema C4ISR se ha convertido en un tema de investigación en el campo militar (Fu, Luo, Luo, & Liu, 2016).

El marco de referencia arquitectónico de *The Open Group* (TOGAF) agiliza y simplifica la definición de la arquitectura empresarial a través un estudio completo de la solución a implementarse para asegurar el crecimiento de una organización en respuesta a las necesidades del negocio. TOGAF trabaja sobre los cuatro dominios de la arquitectura empresarial: Arquitectura de Negocio, Arquitectura de Información, Arquitectura de Aplicaciones y Arquitectura de Tecnología. El componente principal de la estructura TOGAF es el método ADM (*Architecture Development Method*) que realiza un análisis desde la visión general del negocio hasta la parte operativa , y en cada fase establece un desarrollo de acuerdo a las características de la organización con una representación lógica de los componentes (Alba Núñez, 2017). Este ADM está conformado por 8 fases que se organizan de modo repetido y cíclico desde la fase A hasta la H (Ledesma Alvear, 2017).

El modelo arquitectónico “4+1” de Kruchten es un modelo de vistas diseñado por el profesor Philippe Kruchten, en el que se definen 4 vistas de la arquitectura del software. La vista de Escenarios es obligatoria cuando se trabaja el modelo “4+1” vistas ya que todos los elementos de la arquitectura se derivan de los requerimientos que ahí se

presentan. En la vista Lógica se representa la funcionalidad que el sistema proporcionará a los usuarios finales. En la vista de Desarrollo se muestran todos los componentes físicos del sistema, así como las conexiones que conforman la solución (incluyendo los servicios) (Jaramillo-Morillo, Solarte, & Ramírez González, 2017). En la vista de Proceso se explican los procesos del sistema y como se comunican, se enfoca en el comportamiento del sistema en tiempo de ejecución. En la vista Física se representa el sistema desde la perspectiva de un ingeniero de sistemas, describe las conexiones físicas entre los componentes (Hernández Hernández, 2016).

La Arquitectura de Referencia para la Internet de las Cosas (IoT ARM, siglas en inglés) propuesta por el proyecto IoT-A proporciona un marco de referencia que provee comprensión más clara de las características de los sistemas de la IoT y del comportamiento con las posibles interacciones con dichos sistemas (Bauer, Boussard, Bui, & Carrez, 2013). Abarca completamente los problemas de seguridad y privacidad, así como a la escalabilidad y la interoperabilidad entre otros temas. Durante la fase de diseño e implementación del cuerpo de la arquitectura se trabaja con las vistas arquitectónicas. La vista Funcional se enfoca en la descomposición funcional, interacción e interfaces. En la primera etapa de la vista Funcional, los requerimientos se asignan a los diferentes Grupos de funcionalidad del Modelo Funcional IoT; luego se forman grupos de requisitos de funcionalidad similar y un componente funcional para estos requisitos definidos. Por último, los componentes funcionales son refinados después de la discusión con los paquetes de trabajo técnicos. La vista de Información va dirigida a la jerarquía de la información, semántica, procesamiento de información y flujo de información (Pérez, Vargas, & Escobar, 2014). En (Martínez Varsi & Piffaretti Correa, 2017) se explica que la vista de Información describe la forma en que la arquitectura almacena, manipula, administra y distribuye información, o sea, define el flujo de datos entre los diferentes componentes funcionales descritos previamente. En este trabajo se plantea además para IoT ARM la vista de Operación y Despliegue, en ella se describe el ambiente en el que el sistema es desplegado, incluyendo la captura de las dependencias del sistema en su ambiente en tiempo de ejecución. Esta vista discute cómo trasladar el sistema como componentes funcionales aislados, a la selección de diferentes tecnologías de disponibles en IoT para construir el sistema y su comunicación. Identifica tecnologías para los tres principales grupos de elementos del Modelo de Dominio de IoT: los dispositivos, recursos y servicios, teniendo cada uno un problema de despliegue diferente y sugiriendo actividades operativas diferentes (Martínez Varsi & Piffaretti Correa, 2017).

Revista Cubana de Ciencias Informáticas
Vol. 13, No. 1, Enero-Marzo, 2019
ISSN: 2227-1899 | RNPS: 2301
Pág. 143-157
http://rcci.uci.cu

La Arquitectura de Software de Referencia para Objetos Inteligentes en Internet de las Cosas se enfoca a sistemas embebidos, arquitectura en capas basada en el modelo propuesto en (Patterns, 2009). La arquitectura consta de cuatro niveles: dos capas superiores (Presentación y Sistemas Externos), Capa de servicios, Capa de lógica de negocio y Capa de recursos virtuales, recursos físicos y acceso a datos. Posee tres vistas arquitectónicas, haciendo referencia al modelo 4+1 de Kruchten. La vista Lógica representa un soporte a los requerimientos funcionales, es decir, lo que el objeto inteligente debe proveer a los usuarios en términos de servicios. La vista de Procesamiento representa el flujo de datos en el procesamiento de datos del objeto inteligente especificando los algoritmos que se utilizan para ofrecer las funcionalidades requeridas. La vista Física representa cómo se distribuyen los componentes entre los distintos nodos del sistema, es decir, muestra cómo se ubica cada parte del software en un nodo de forma tal que se mapeen software y hardware. Se optó por omitir las vistas de Desarrollo y de Escenarios, dado que ambas dependen exclusivamente de los requerimientos específicos para el objeto inteligente, elementos que no se contemplan por ser una arquitectura de referencia (Segura, 2016).

Las Arquitecturas de Referencias de Software consideradas, coinciden que en el 100% de los trabajos se utiliza la vista arquitectónica como elemento para la descripción de la arquitectura. Según (Ochoa, 2011) la mayoría de los marcos de trabajo y estrategias consultadas reconocen las vistas como la estructura central para la formalización y descripción de las arquitecturas. Aun cuando algunos modelos incluyen procedimientos, estándares de modelado o incluso configuraciones de herramientas, lenguajes y procedimientos formales, vuelve a resultar la vista arquitectónica el epicentro de la especificación en los modelos o métodos de desarrollo arquitectónico como artefactos rectores de este proceso y soporte formal para la comunicación de los involucrados en un proceso de desarrollo de software.

Con la formalización de una arquitectura a través de las vistas arquitectónicas, se hace más sencilla la comunicación entre el equipo de trabajo por el empleo de lenguaje natural y semiformal; sin embargo, esta forma de trabajo limita el uso de herramientas automatizadas que pueden interpretar, por ejemplo: lógica descriptiva para representar conocimiento de un dominio dado y bien estructurado. Además, impacta en el proceso de reutilización del diseño arquitectónico, al comparar varias arquitecturas de referencias en un lenguaje no formal, se pueden introducir errores pues la decisión se basa en la experticia del arquitecto y no en un análisis comparativo por alguna herramienta atendiendo a restricciones definidas.

Conclusiones

La investigación permitió profundizar en los elementos conceptuales más significativos para definir las arquitecturas de referencias de software. Los conceptos más relevantes (Componente y Conector) están incluidos en el libro basado en el estándar de la IEEE 1471 de (Rozanski & Woods, 2012), donde se plantea un modelo conceptual para la definición de Arquitectura de Software. Del estudio se obtuvo que la mayoría de los modelos revisados utilizan la vista arquitectónica como el elemento principal para representar las arquitecturas de referencias de software, característica que limita el uso de herramientas automatizadas por no hacer uso de un lenguaje formal. Los hallazgos relacionados con las arquitecturas de referencias de software se tendrán en cuenta para el desarrollo de una propuesta que permita representar el conocimiento arquitectónico relativo a las decisiones arquitectónicas y su razonamiento.

Referencias

Abbasi, M. A., Butt, R., & Anjum, T. M. (2016). Comparative Analysis of Software Architecture Documentation and Architecture Languages. Paper presented at the Computer Science and Engineering (APWC on CSE), 2016 3rd Asia-Pacific World Congress on.

Alba Núñez, J. M. (2017). Modelo de referencia de negocio basado en TOGAF para la Universidad Técnica Particular de Loja.

Bachmann, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., . . . Wallnau, K. (2000). Volume II: Technical concepts of component-based software engineering: Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon Software Engineering Institute.

Balouki, A., Sbihi, M., & Balouki, Y. (2015). RM-ODP: A framework for Mechatronics systems. Paper presented at the Research and Education in Mechatronics (REM), 2015 16th International Conference on.

Bass, L., Clements, P., & Kazman, R. (2003). Software architecture in practice: Addison-Wesley Professional.

Bauer, M., Boussard, M., Bui, N., & Carrez, F. (2013). Project deliverable D1. 5 final architectural reference model for IoT. IoT-A Project-UniS, Tech. Rep., Jul.

Carpio Encalada, J. M., & Cango, R. V. F. (2016). Desarrollo de un sistema web de administración y visualización de alertas en tiempo real con notificación vía mensaje de texto y una aplicación móvil con geolocalización de emergencias médicas para la Cruz Roja-Loja.

Clements, P. C. (1996). A survey of architecture description languages. Paper presented at the Proceedings of the 8th international workshop on software specification and design.

España, S., & Fernando, H. (2016). Documentación y análisis de los principales frameworks de arquitectura de software en aplicaciones empresariales. Facultad de Informática.

Fu, J., Luo, A., Luo, X., & Liu, J. (2016). A Component Business Model-based Approach for Business Architecture Integration of C4ISR System. Paper presented at the Information Science and Control Engineering (ICISCE), 2016 3rd International Conference on.

Hernández Hernández, M. J. (2016). Herramienta de soporte al desarrollo de competencias básicas del ciudadano digital.

Hernández, Y. J., Velasco-Elizondo, P., & Benítez-Guerrero, E. (2016). Evaluando Adecuación Funcionaly Usabilidad en Herramientas de Composición desde la Perspectiva del Usuario Final. RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação(17), 96-114.

Hirsch, D., Kramer, J., Magee, J., & Uchitel, S. (2006). Modes for software architectures. Paper presented at the European Workshop on Software Architecture.

Jaramillo-Morillo, D., Solarte, M., & Ramírez González, G. (2017). Estrategia de seguimiento a las actividades de aprendizaje de los estudiantes en cursos en línea masivos y privados (MPOC) con reconocimiento académico en la Universidad del Cauca.

Ledesma Alvear, J. C. (2017). Frameworks de arquitectura empresarial. Facultad de Informática.

Martínez, J., & Pino, F. J. (2016). Definición de un Modelo de Calidad de Servicios Soportado por Tecnologías de la Información (TI). Publicaciones e Investigación, 10, 49-67.

Martínez Varsi, N., & Piffaretti Correa, C. G. (2017). Diseño de una Arquitectura de Software orientada a la Internet de las Cosas a partir de un Modelo Arquitectónico de Referencia.

Menéndez, D. B. (2015). Arquitectura de Referencia para una Plataforma Integral de Telecomunicaciones Unificadas, ARPlaTel-U. (Tesis de maestría), Universidad de las Ciencias Informáticas.

Milosevic, Z., & Bond, A. (2016). Digital health Interoperability frameworks: use of RM-ODP standards. Paper presented at the Enterprise Distributed Object Computing Workshop (EDOCW), 2016 IEEE 20th International.

Ochoa, R. L. (2011). ¨Modelo de referencia para el desarrollo arquitectónico de sistemas de software en dominios de gestión¨. (Máster en Ciencias Técnicas), Universidad de las Ciencias Informáticas.

Ozkaya, M., & Kloukinas, C. (2013). Are we there yet? Analyzing architecture description languages for formal analysis, usability, and realizability. Paper presented at the 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA).

Patterns, M. (2009). Microsoft® Application Architecture Guide: Microsoft Press.

Pérez, L. A. J., Vargas, W. A. V., & Escobar, N. F. V. (2014). Estado del arte de las arquitecturas de internet de las cosas (iot).

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, 64, 1-18.

Pressman, R. (2010). Ingeniería del Software UIn enfoque práctico, Septima edición ed: McGraw-Hill.

Quecan Rueda, C. L., & Pulido Vásquez, J. P. “QCDL4” Extensión de Qgis para la Generación de Linderos Prediales.

Reynoso, C. (2004). Introducción a la Arquitectura de Software. Universidad de Buenos Aires, 33.

Rozanski, N., & Woods, E. (2012). Software systems architecture: working with stakeholders using viewpoints and perspectives: Addison-Wesley.

Segura, A. A. (2016). Arquitectura de software de referencia para objetos inteligentes en internet de las cosas. Revista Latinoamericana de Ingenieria de Software, 4(2), 73-110.

Sepúlveda, C. A. O. (2017). Arquitectura de línea base para el framework Ontoconcept. (Maestría en Ingeniería de Sistemas), UNIVERSIDAD TECNOLÓGICA DE PEREIRA.

Sommerville, I. (2011). Ingeniería de software novena edición: México: Pearson.

Velasco, C. C., Linero, D. J. M. T., de Lenguajes, A., & Sánchez, D. E. P. (2000). Un lenguaje para la especificación y validación de arquitecturas de software. Málaga.

Zarvić, N., & Wieringa, R. (2014). An integrated enterprise architecture framework for business-IT alignment. Designing Enterprise Architecture Frameworks: Integrating Business Processes with IT Infrastructure, 63, 9.