risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Revista Ibérica de Sistemas y Tecnologías de Información

Recebido/Submission: 21/05/2019
Aceitação/Acceptance: 14/09/2019

Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube

María Julia Blas[1,2], Horacio Leone[1,2], Silvio Gonnet[1,2]

mariajuliablas@santafe-conicet.gov.ar, hleone@santafe-conicet.gov.ar, sgonnet@santafe-conicet.gov.ar

[1] INGAR – Instituto de Desarrollo y Diseño, CONICET-UTN, Santa Fe, CP: 3000, Santa Fe, Argentina.

[2] Universidad Tecnológica Nacional –Regional Santa Fe, Santa Fe, CP: 3000, Santa Fe, Argentina.

DOI: 10.17013/risti.35.1–17

Resumen: Este trabajo presenta un entorno de diseño integral que permite formular diseños de arquitecturas de software destinadas a la representación de aplicaciones web. Este entorno abstrae los principales problemas identificados a nivel de diseño, planteando módulos que ayudan al arquitecto en la elaboración de diseños de calidad. Para esto, utiliza como base un metamodelo de componentes arquitectónicos que identifica un conjunto de elementos comúnmente utilizados en dichas arquitecturas. Sobre el modelo se construye una herramienta de instanciación gráfica que se complementa con la verificación de patrones de diseño a fin de garantizar su correcta aplicación.

Palabras-clave: aplicación web; diseño de arquitectura; patrones de diseño.

Modeling and Verification of Software Architecture Design Patterns for Cloud Computing Environments

Abstract: This work presents a design environment that allows building software architecture designs for web applications. The environment is designed to solve some of the main problems identified at architectural level by proposing a set of modules that help to develop quality architectures. Its structure is based on a metamodel of architectural components that identifies a set of elements commonly used in web application architectures. Also, a graphical software tool is built using this model as a support mechanism. Such tool is completed with a verification software module that ensures the correct application of well-known web design patterns.

Keywords*:* web application; architecture design; design pattern.

1. Introducción

Las funcionalidades de un producto de software son ortogonales respecto a los atributos de calidad (Bass, Clements & Kazman, 2012). Cuando se evalúa la calidad de forma

continua durante el desarrollo se intenta *garantizar que el producto tendrá el efecto requerido en los contextos de interés*. Sin embargo, la intrínseca complejidad del software dificulta la determinación de la calidad (Pressman, 2010).

De acuerdo con (Albin, 2003), entre los problemas que dificultan la especificación, diseño, medición y prueba de los atributos de calidad de un producto se destacan: lenguaje inadecuado para expresar los requerimientos de calidad, dificultades para generar diseños basados en atributos de calidad y falta de documentación en el diseño y en los patrones arquitectónicos que brindan soporte a las posibles configuraciones. Esta situación se complejiza al intentar aplicar los conceptos y técnicas existentes sobre plataformas más novedosas basadas en modelos de ejecución modernos; por ejemplo, el esquema de *computación en la nube* (CC).

En este contexto, la adopción de la tecnología de CC no es una tarea sencilla (Palos-Sánchez, Arenas-Márquez & Aguayo-Camacho, 2017). El CC (en comparación con los productos de software tradicionales) obliga a formular cambios en las normas y estándares, las capacidades de infraestructura y la especificación de las arquitecturas de software que brindan soporte a las aplicaciones y/o servicios (Silva, Verdúm, Espinoza, Hurtado & Poma, 2015). Teniendo en cuenta que la determinación del conjunto de propiedades que define un nivel adecuado de calidad para un sistema de software es una pregunta altamente dependiente del contexto (Kitchenham & Pfleeger, 1996); el estudio, entendimiento y análisis de la calidad en CC se ve afectado por la naturaleza de dicho entorno. Aunque existe una amplia variedad de patrones útiles para el diseño de software (Garlan & Shaw, 1994; Pahl, Giesecke & Hasselbring, 2009), al intentar trasladar estos patrones a los servicios de computación en la nube se presentan dos problemas derivados de la falta de madurez del área: *i)* los patrones de CC aún no se encuentran totalmente establecidos; y *ii)* no existe una técnica de representación que facilite la comprensión de las arquitecturas de software basadas en CC. Esta situación dificulta la labor de los arquitectos, dando como resultado una ralentización de la tarea de diseño. Luego, resulta necesario brindar toda la ayuda posible a los arquitectos de CC a fin de aumentar su productividad, contribuyendo a la formulación de diseños adecuados en base a distintas propiedades de calidad.

Este trabajo propone una herramienta basada en un metamodelo de patrones de arquitecturas de software en la nube que facilita la tarea de modelado y, además, posibilita la verificación semiautomática de los patrones aplicados a fin de garantizar su correcta instanciación. Complementariamente, se sientan las bases necesarias para la definición de un *entorno de diseño integrado* que incluya guías de diseño basadas en el conjunto de componentes y conexiones propuestos junto con sugerencias inteligentes derivadas de las decisiones arquitectónicas adoptadas por el usuario a lo largo de la construcción del diseño.

La Sección 2 introduce los problemas detectados en el proceso de diseño de arquitecturas para entornos de CC, los cuales han sido tomados como principios fundamentales para la formulación de la propuesta. La Sección 3 presenta la estructura del entorno de diseño propuesto como parte del trabajo, incluyendo una breve descripción de cada uno de los módulos que lo componen. La Sección 4 describe el metamodelo arquitectónico diseñado como vocabulario base de este entorno de diseño sobre el cual, en la Sección 5, se introduce el módulo de verificación de patrones de diseño (que contribuye a la

evaluación de las arquitecturas de software instanciadas). Finalmente, la Sección 6 se encuentra destinada a las conclusiones y trabajos futuros que se desprenden del entorno propuesto.

2. Diseño de Arquitecturas para Servicios de Software

El diseño arquitectónico es un proceso creativo en el que se intenta establecer una organización de los componentes del sistema de software que satisfaga tanto los requerimientos funcionales como los requerimientos no funcionales (Sommerville, 2005). Luego, al construir una arquitectura de software es necesario seleccionar los principios de trabajo adecuados con el objetivo de lograr un balance apropiado de las propiedades deseadas sobre el producto de software bajo diseño. Tales propiedades se desprenden de un conjunto de requerimientos que compiten entre sí. De acuerdo con (Bass, Clements & Kazman, 2012), la *arquitectura de un producto de software* comprende los componentes del software, las propiedades visibles de dichos componentes y las relaciones que existen entre ellos. Esta definición es indiferente al hecho de que el diseño formulado sea bueno o malo (es decir, permita o evite que el producto de software bajo desarrollo cumpla con los requerimientos identificados). Por este motivo, las arquitecturas de software deben ser evaluadas a fin de determinar su habilidad para cumplir con los atributos de calidad relevantes del sistema de software bajo desarrollo.

El proceso de evaluación de este tipo de diseños no debe centrarse en la aplicación de una única métrica universal (inexistente), sino que debe cuantificar atributos individuales para, luego, realizar una compensación sobre las métricas obtenidas (Barbacci, Klein, Longstaff & Weinstock, 1995). En este sentido, el arquitecto de software debe: *i)* contar con la habilidad requerida para evaluar cuantitativamente los atributos de calidad sobre un diseño funcional dado, y *ii)* en base al resultado de esta evaluación, realizar una compensación de características que le permita alcanzar el mejor comportamiento posible sobre el sistema de software bajo análisis.

Luego, la formulación de diseños de arquitecturas de software se caracteriza por su complejidad, donde el punto de partida es un conjunto de objetivos que no se encuentra claramente definido, cuya especificación se obtiene a medida que se lleva a cabo el proceso de solución (Roldán, Gonnet & Leone, 2016).

De acuerdo con el Instituto Nacional de Normas y Tecnología (NIST por sus siglas en inglés, National Institute of Standards and Technology), el paradigma de CC es *un modelo que habilita acceso a Internet bajo demanda de forma conveniente y generalizada a fin de compartir un conjunto de recursos de computación configurables que pueden ser asignados y entregados de forma rápida con un esfuerzo marginal reducido o con mínima interacción con el proveedor del servicio* (Mell & Grance, 2011). En este contexto, la Figura 1 presenta una primera aproximación a la estructuración arquitectónica de aplicaciones web basada en dos niveles de abstracción: *infraestructura* y *aplicación*. Las funcionalidades se encuentran alojadas en la capa de aplicación mientras que los recursos (sobre los cuales se ejecutan dichas funcionalidades) quedan encapsulados en la capa de infraestructura. De acuerdo con este diagrama de alto nivel, los *usuarios* utilizan los servicios de software por medio de un acceso indirecto a aplicaciones remotas alojadas en servidores externos pertenecientes a proveedores de infraestructura.

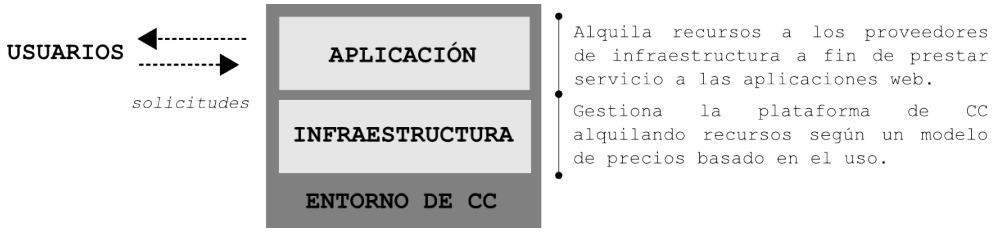

Figura 1 – Descomposición arquitectónica de alto nivel del diseño de aplicaciones web basadas en entornos de CC.

En el caso del diseño arquitectónico asociado a las aplicaciones web, el proceso creativo requiere de arquitectos especializados en el entorno de CC que sean capaces de formular soluciones factibles no sólo de ser implementadas en base a módulos de software sino también susceptibles de ser desplegadas sobre infraestructuras en la nube (de forma tal que brinden solución a los requerimientos no funcionales). Luego, la tarea de diseñar arquitecturas de software para aplicaciones de CC requiere de arquitectos con las habilidades necesarias tanto para identificar estas posibles situaciones como así también para solucionar los problemas derivados de los diferentes dominios.

Frecuentemente el arquitecto carece de información cualitativa y/o cuantitativa de utilidad para el soporte de la evaluación y selección de alternativas de diseño parciales (Kazman, Gagliardi & Wood, 2012). En muchos casos, esta situación sumada al desconocimiento propio de la "caja negra" de los entornos de CC, dificulta el estudio de la arquitectura del producto. En este contexto, las estrategias de evaluación deben basarse en una combinación de técnicas y elementos preexistentes que contribuyan al análisis del diseño arquitectónico propuesto.

2.1. Dificultades de la Tarea de Diseño de Arquitecturas Web: Problemas de Diseño y Conocimientos Requeridos

Tal como se ha establecido con anterioridad, en el área de diseño de arquitecturas de software web existen dos problemas derivados de la falta de madurez en la disciplina:

- *Los patrones de diseño orientados a CC aún no están totalmente establecidos*: Dada la variedad de enfoques que pueden utilizarse para el diseño de este tipo de arquitecturas, muchos de los patrones existentes refieren a soluciones aplicables en relación a la tecnología de la información que brinda soporte al nivel de infraestructura (no de aplicación).
- *Las técnicas de representación utilizan enfoques híbridos que combinan elementos pertenecientes a distintos niveles de definición*: Las estrategias de representación arquitectónica mezclan elementos de infraestructura con componentes de software de aplicación. Tal combinación no sólo dificulta la identificación de los elementos requeridos para cumplir los objetivos de diseño, sino que, además, es susceptible de generar confusión y ambigüedad a diseñadores inexpertos.

Frecuentemente, a fin de evitar los conflictos derivados de la falta de pericia al momento de generar un diseño, los arquitectos trabajan en base a arquitecturas genéricas. Una *arquitectura genérica* provee una estructura para una familia de problemas, encapsula propiedades inherentes a un dominio específico y guía el proceso de diseño hacia la obtención de la calidad. De esta manera, una "buena" estructura: *i)* reduce la posibilidad de obtener un diseño inflexible, y *ii)* permite a los diseñadores concentrarse en los aspectos importantes del sistema bajo desarrollo. Por el contrario, una "mala" estructura fuerza al diseñador a tomar decisiones que limitan (a futuro) las posibilidades de modificar o alterar el diseño a un costo razonable.

En este contexto, en términos generales existen dos conceptos asociados a la aplicación de *arquitecturas genéricas*: *patrones de diseño* y *decisiones arquitectónicas*. Sin embargo, en el caso de las arquitecturas de software basadas en servicios web, la formulación del diseño a nivel de aplicación debe contemplar también el despliegue de sus componentes sobre la infraestructura subyacente. Luego, la aplicación de las *arquitecturas genéricas* basadas en *patrones de diseño* y *decisiones arquitectónicas* se ve condicionada por las *condiciones de despliegue en infraestructura*.

Patrones de Diseño Arquitectónico

Un *patrón de diseño arquitectónico* define una familia de sistemas en base a un esquema de organización estructural (Garlan & Shaw, 1994). Bajo esta perspectiva, los patrones arquitectónicos pueden ser vistos como los bloques constructivos básicos de las arquitecturas de software. Esto es, definen y organizan un vocabulario de elementos basado en un conjunto de componentes y conectores asociados a un grupo de restricciones que indican la forma en la cual tales elementos pueden ser combinados. El objetivo de este vocabulario es la formulación de diseños arquitectónicos que garanticen una adecuada composición de los elementos que lo conforman según el esquema de organización seleccionado. Esta selección no debe realizarse de forma arbitraria, sino que siempre se encuentra relacionada con los requerimientos no funcionales de la aplicación bajo desarrollo. De acuerdo con esta perspectiva, los patrones de diseño arquitectónico describen pares de elementos *{problema; solución}* con el objetivo de brindar soluciones abstractas a problemas recurrentes en un dominio específico.

Las arquitecturas de sistemas de software complejos suelen basarse en la aplicación de más de un patrón de diseño (quedando definidas en base a una composición de estos últimos). Este es el caso de las arquitecturas de aplicaciones web, en donde la construcción del diseño arquitectónico implica vincular múltiples componentes haciendo uso de diversos esquemas de organización. En este sentido, los patrones de diseño de aplicaciones web describen soluciones abstractas a problemas recurrentes en el dominio de entornos de CC a fin de capturar conocimiento independiente de los proveedores de servicios de infraestructura. Estos patrones ayudan a determinar la forma en la cual deben ser distribuidos los distintos tipos de componentes de una aplicación web. Luego, la identificación de los patrones adecuados es una tarea importante que debe ser llevada a cabo por el arquitecto a cargo de la construcción del diseño. Tal responsabilidad depende directamente de su pericia y experiencia en el dominio de trabajo vinculado a la aplicación bajo desarrollo.

No todos los patrones de CC son nuevos. Muchos de los patrones existentes para las arquitecturas de software tradicionales han sido adaptados para su aplicación en contextos de aplicaciones web basadas en entornos de CC. Sin embargo, no todos los patrones de diseño existentes abstraen correctamente los elementos que conforman cada uno de los niveles arquitectónicos requeridos (esto es, software e infraestructura). En este contexto, el conjunto de patrones de diseño propuesto en (Fehling, Leymann, Retter, Schupeck & Arbitter, 2014) presenta una estrategia de organización basada en un conjunto de elementos el cual ha sido definido de acuerdo a una clasificación de componentes en términos de categorías (Tabla 1). Cada categoría refiere a un nivel de trabajo, a saber: *cloud application* (esto es, el nivel de aplicación) y *cloud runtime environment* (esto es, el nivel de infraestructura).

Luego, los patrones identificados en los distintos niveles arquitectónicos constituyen una esquematización de diseño elemental que permite agrupar múltiples estructuras con el objetivo de generar diseños aplicables a productos de software específicos.

Nivel	Categoría	Descripción
Cloud Application	Hybrid / Native Cloud Applications	Patrones que describen la forma en la cual se distribuyen los componentes de aplicación en base al tipo de entorno.
	Fundamental Architecture Styles	Patrones que los arquitectos y desarrolladores deben tener en cuenta al construir una aplicación en la nube.
	Application Components	Patrones centrados en la especificación del manejo del estado de componentes de aplicación en base al diseño.
	Multi-tenancy	Patrones de aplicaciones en la nube que se ofrecen como servicio a múltiples clientes.
	Cloud Integration	Patrones útiles para aquellas aplicaciones que se distribuyen entre diferentes entornos y/o deben integrarse con otras aplicaciones alojadas en distintos entornos.
	Application Management	Patrones que describen formas alternativas de manejar automáticamente las aplicaciones de nube haciendo uso de múltiples instancias de componentes.
Cloud Runtime Environment	Cloud Service Models	Patrones basados en modelos de servicios según las capas de la pila de aplicación sobre las cuales proporciona recursos de tecnología de la información.
	Cloud Deployment Types	Patrones de despliegue que se diferencian principalmente por los grupos de usuarios que acceden a la nube y por el grado en que los recursos se comparten entre los clientes.
	Cloud Environments	Patrones que caracterizan los ambientes creados en base a diferentes modelos de despliegue de acuerdo a los componentes requeridos.
	Processing / Communication / Storage Offerings	Patrones que refieren a diferentes ofertas alojadas en la nube que proporcionan una determinada funcionalidad a los clientes en base a un comportamiento predefinido.

Tabla 1 – Clasificación de los patrones arquitectónicos propuestos por (Fehling, Leymann, Retter, Schupeck & Arbitter, 2014).

Decisiones Arquitectónicas

Una *decisión arquitectónica* es una descripción del conjunto de agregados, eliminaciones y modificaciones realizadas sobre una arquitectura de software, el motivo de que estos cambios tengan lugar y las reglas de diseño junto con las respectivas restricciones y especificaciones adicionales que (posiblemente de forma parcial) logran cubrir uno o más de los requerimientos de la arquitectura (Jansen & Bosch, 2005). Luego, en términos generales, una decisión arquitectónica es la salida que se genera durante un proceso de diseño arquitectónico en cualquier fase de trabajo (esto es, como parte de la construcción inicial o en fases de evolución) de un sistema de software. En este contexto, aunque no se formalizan, las decisiones tomadas por los arquitectos se consideran basadas en sus propias experiencias (Carignano, Gonnet & Leone, 2019).

Múltiples autores plantean que las decisiones arquitectónicas son el núcleo de las arquitecturas de software (Jansen & Bosch, 2005; Tyree & Akerman, 2005). Esto se debe a que las decisiones de diseño refieren al dominio de aplicación del sistema de software bajo desarrollo, los patrones arquitectónicos utilizados para la formulación de su estructura, los componentes definidos como parte del diseño, la infraestructura seleccionada para su ejecución y todos los aspectos adicionales necesarios para satisfacer los requerimientos relevados (tanto funcionales como no funcionales). Dada la variabilidad que existe en las arquitecturas de CC, la documentación de las decisiones tomadas por los arquitectos en diferentes proyectos sirve como descriptor de las habilidades requeridas y del conocimiento implícito que se formula por detrás de estas determinaciones.

En este contexto, a fin de absorber el dinamismo propio con el cual crecen los diseños arquitectónicos de entornos de CC, es fundamental que la introducción de cambios arquitectónicos se realice de forma sistemática. De esta manera, se evita la erosión en el diseño arquitectónico y la pérdida de información vital para la comprensión del diseño obtenido, tanto sea en esta instancia de desarrollo como así también en instancias de trabajo futuras evidenciadas en proyectos de desarrollo similares (Jansen & Bosch, 2005).

Condiciones de Despliegue en la Infraestructura

Tal como se ha especificado con anterioridad, al formular una arquitectura para entornos de CC se deben contemplar dos niveles de diseño: aplicación e infraestructura. A nivel de aplicación se debe tener en cuenta que el diseño debe ser implementado sobre la infraestructura subyacente. En este sentido, el arquitecto a cargo debe contemplar no sólo el conjunto de componentes requeridos con el objetivo de cumplir los requerimientos funcionales (junto con sus relaciones), sino también la forma en la cual estos componentes de aplicación deben ser desplegados sobre el nivel de infraestructura. En cualquier caso, las evaluaciones que quieran realizarse en base al diseño arquitectónico completo deben necesariamente contemplar este aspecto ya que, tanto su funcionamiento como su rendimiento se ven condicionados por el contexto de ejecución definido para la aplicación. En la mayoría de los casos, este contexto de infraestructura es contratado a un proveedor externo.

En la actualidad existen muchos proveedores que brindan múltiples tipos de servicios para la definición de la infraestructura de las aplicaciones web. La principal característica

de estos servicios es que el usuario paga únicamente por el consumo que realiza. Luego, al contratar el servicio, el desarrollador sólo debe preocuparse por definir los tipos de instancias adecuadas que le permitan lograr el cumplimiento de los requerimientos funcionales y de calidad asociados a la aplicación de software. Sin embargo, esta característica depende de la experiencia del arquitecto a cargo del diseño en relación al uso de los diferentes proveedores de infraestructura.

2.2. Evaluación de Arquitecturas de CC según su Diseño

Las arquitecturas de software han evolucionado de simples representaciones estructurales a esquemas centrados en decisiones (Kruchten, Capilla & Dueñas, 2009). En este sentido, tanto los *patrones de diseño* estructurales como las *decisiones de diseño* se complementan entre sí. Sumando la *dependencia de la infraestructura*, es evidente que en un entorno de CC la construcción de arquitecturas de software se complejiza al considerar todos los aspectos requeridos para la obtención de un diseño apropiado.

En este contexto, se tiene que:

- Los *patrones de diseño* ayudan al arquitecto a resolver problemas frecuentes en términos de composiciones utilizadas como esquemas de trabajo, cuyo punto de partida es un conjunto predefinido de componentes y conexiones válidas.
- Las *decisiones arquitectónicas* se basan en la evolución de las arquitecturas como consecuencia de cambios y/o mejoras en su definición. Estas decisiones deben documentarse según los componentes y/o conexiones modificados a fin de facilitar tanto la recuperación de las soluciones aplicadas como así también analizar su impacto sobre el diseño.
- Las *dependencias de infraestructura* se formulan vinculando componentes arquitectónicos de nivel de software a componentes arquitectónicos de nivel de infraestructura (los cuales, en la práctica, corresponden a ofertas de proveedores).

Luego, estos tres aspectos comparten como problema base la *identificación del conjunto de componentes y conectores válidos para la especificación de arquitecturas de CC*. Una definición apropiada de los elementos disponibles para la formulación de los diseños arquitectónicos contribuye al proceso de diseño ya que, de forma genérica, los arquitectos pueden aplicar lineamientos estándares referidos a patrones de diseño. En base a los lineamientos aplicados, es posible documentar las decisiones arquitectónicas adoptadas a fin de garantizar la adecuación del conjunto de modificaciones y su impacto en términos de los patrones involucrados. Además, conociendo los tipos de componentes disponibles, es posible brindar información asociada a los proveedores de servicios de infraestructura que mejor se ajustan al diseño formulado.

En la siguiente sección se presenta la estructura del entorno de diseño propuesto con el objetivo de brindar a los arquitectos un mecanismo de soporte integral que ayude a aumentar su productividad en el desarrollo de arquitecturas de CC. Este trabajo se centra en la definición del vocabulario de componentes y conectores básico que brinda soporte a la construcción de arquitecturas web, dando lugar a un uso apropiado de los patrones de diseño identificados en el nivel de aplicación. Los patrones relacionados al nivel de infraestructura son conceptualizados como servicios contratados a terceros.

3. Entorno de Diseño Integrado para Arquitecturas Web

La Figura 2 esquematiza el conjunto de módulos identificados como parte del entorno de diseño propuesto.

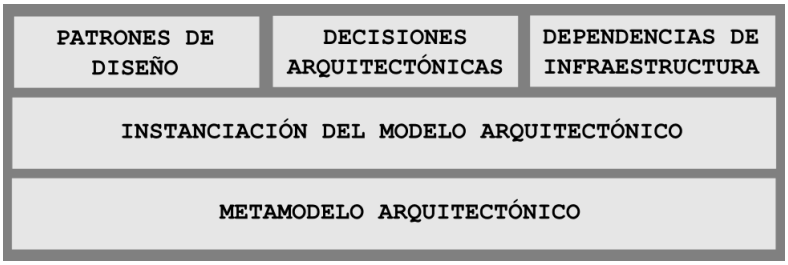

Figura 2 – Arquitectura del entorno definida en términos de módulos de software.

Tal como se ha enunciado con anterioridad, la base de trabajo consiste en un módulo conceptual denominado *metamodelo arquitectónico* que define el vocabulario de elementos disponibles para la creación de arquitecturas de CC. Un metamodelo puede definirse como "un modelo de un modelo". Esto quiere decir que las instancias de este tipo especial de modelos corresponden a un nuevo tipo de modelo que, a su vez, puede ser instanciado (Atkinson & Kuhne, 2003). En este caso, el *metamodelo arquitectónico* posibilita la *instanciación de modelos de arquitecturas* en base a un conjunto predefinido de componentes y conectores arquitectónicos.

En base a las instanciaciones que se producen como resultado de la utilización de los componentes y conectores arquitectónicos identificados, se definen tres nuevos módulos que actúan como complementos en la especificación de arquitecturas de CC, a saber:

- Módulo *patrones de diseño*: Verifica la correcta aplicación del conjunto de patrones de diseño propuesto en (Fehling, Leymann, Retter, Schupeck & Arbitter, 2014) sobre el modelo de arquitectura instanciado, dando lugar a una evaluación del diseño.
- Módulo *decisiones arquitectónicas*: Aplica un modelo de representación del conocimiento durante la evolución del diseño basado en la trazabilidad de los elementos arquitectónicos involucrados en las decisiones tomadas por el arquitecto.
- Módulo *dependencias de infraestructura*: Utiliza la información provista por los proveedores de infraestructura a fin de asistir al arquitecto en el proceso de selección y contratación de los servicios requeridos para el despliegue de los componentes de software (incluidos en el modelo) sobre el entorno de CC.

Luego, el entorno propuesto abarca el proceso de definición y evolución de la arquitectura en base a un único conjunto de elementos que posibilitan la integración de conceptos.

4. Metamodelo de Componentes Arquitectónicos

En virtud de obtener un conjunto acotado de elementos que facilite la definición de arquitecturas de CC a nivel de componentes de software, el metamodelo diseñado se basa en el estudio de múltiples patrones de diseño de CC. De esta manera, los componentes

y conectores identificados como parte del modelo posibilitan la instanciación de arquitecturas factibles de ser implementadas en entornos de CC.

4.1. Tipos de Componentes y Vínculos Incluidos en el Metamodelo

Del estudio de los patrones de diseño de CC se identifican tres clases de componentes arquitectónicos:

- *Componentes funcionales*: Componentes que definen responsabilidades asociadas a funciones específicas. Deben ser utilizados para modelar el comportamiento de los *componentes de aplicación no definidos*.
- *Componentes de aplicación*: Componentes que se utilizan para detallar el comportamiento de las aplicaciones web. Se dividen en: *i) Componentes de aplicación definidos*: Elementos comúnmente utilizados en la definición de arquitecturas web, cuyo funcionamiento no varía según el dominio de trabajo. *ii) Componentes de aplicación no definidos*: Elementos específicos del dominio de aplicación asociado al producto de software web, cuyo comportamiento debe modelarse empleando *componentes funcionales*.
- *Componentes de administración*: Componentes que se utilizan para gestionar automáticamente el comportamiento de la aplicación web por medio del monitoreo de los *componentes de aplicación* que la componen.

Por su parte, los vínculos formados entre estos tipos de componentes pueden ser clasificados de acuerdo a tres formatos diferentes:

- *Vínculos de aplicación*: Conexiones establecidas entre componentes de alto nivel (*componentes de aplicación* y *componentes de administración*) con el objetivo de indicar una interacción funcional.
- *Vínculos de flujo*: Conexiones entre *componentes funcionales* que detallan el comportamiento de una función compleja en base a una secuencia de control definida sobre funciones elementales.
- *Vínculos especiales*: Conexiones establecidas entre *componentes de administración* y *componentes de aplicación* (específicos) con el objetivo de gestionar la cantidad de réplicas de componentes arquitectónicos requerida para dar respuesta a la carga de trabajo actual y la asignación de recursos de tecnología de la información.

En este contexto, el metamodelo incluye elementos asociados a ambas categorías (componentes y vínculos). En el caso de los vínculos arquitectónicos, se genera una subdivisión que contempla el nivel de aplicación afectado por la conexión (externo/interno) y el sentido en el cual es aplicada en relación a los componentes que une (unidireccional/bidireccional).

4.2. Especificación del Metamodelo Arquitectónico

La definición del *metamodelo* incluye dos especificaciones complementarias: *i)* una *descripción en Unified Modeling Language (UML)* que define el conjunto de clases, relaciones y atributos necesarios para modelar las arquitecturas de entornos de CC, y *ii)* un *conjunto de restricciones en Object Constraint Language (OCL)* que garantiza la consistencia de los modelos instanciados a partir de los objetos definidos en UML.

El modelo UML se divide en cuatro grupos de descripción, a saber: aplicación en la nube (*cloud application description*), descomposición basada en capas (*layer decomposition description*), componentes arquitectónicos (*architectural components description*) y estado de componentes (*components state description*).

Los elementos de la sección *cloud application description* (Figura 3, recuadro amarillo) permiten especificar la arquitectura del entorno de CC sobre la cual se despliega la aplicación web bajo diseño. Tal especificación se logra instanciando la clase *Representation*. Dicha instancia debe contener una aplicación (*CloudApplication*) y el enfoque de descomposición asociado (*Decomposition*). Ambos elementos modelan aspectos genéricos del CC, por lo que es necesario que sus instancias se relacionen con aspectos específicos de un entorno particular. Por este motivo, las instancias que se utilicen para describir el concepto *Representation* se deben corresponder (según la estrategia de descomposición elegida por el arquitecto) con alguna de las clases que especializan los términos abstractos definidos.

Como la mayoría de las arquitecturas de CC se basan en un modelo de capas, en la sección *layer decomposition description* se presenta su definición (Figura 3, recuadro verde). Por medio de la clase *CloudApplicationDecomposedInLayers* se define su esquema en base a dos elementos estructurales: *RuntimeEnvironment* y *Application*. La clase *RuntimeEnvironment* representa el conjunto de capas inferiores que componen el entorno de CC (la infraestructura), mientras que la clase *Application* modela la capa sobre la cual los arquitectos diseñan el software. Como el objetivo del metamodelo es especificar el conjunto de conceptos y relaciones requeridos para el diseño arquitectónico de aplicaciones web (ya que la infraestructura es conceptualizada como un servicio contratado a uno o más proveedores externos), el modelo profundiza la definición estructural en la clase *Application*. En este caso, la definición del enfoque de distribución

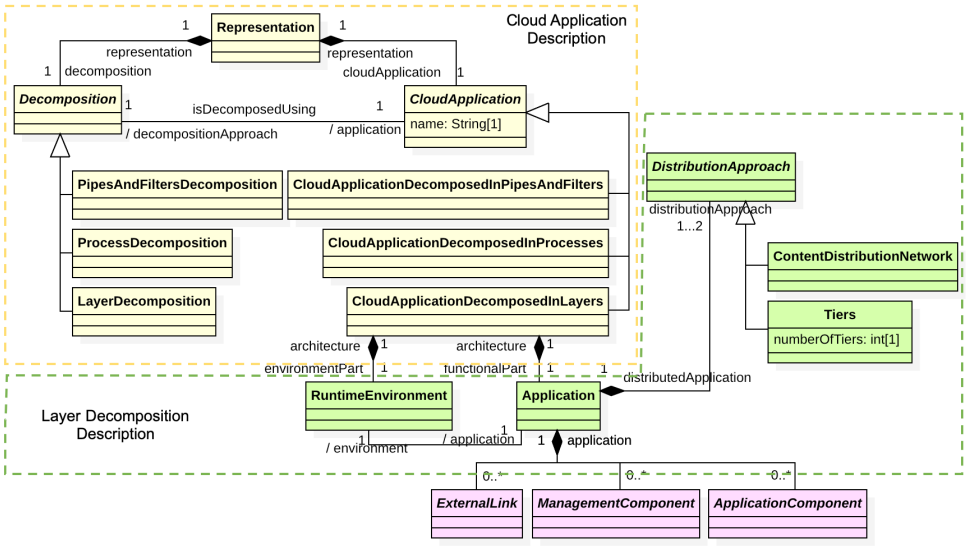

Figura 3 – Metamodelo arquitectónico (partes 1 y 2).

a utilizar sobre los componentes de aplicación a diseñar se indica por medio de la clase *DistributionApproach*.

Los conceptos de la sección *architectural components description* (Figura 4, recuadro rosado) se dividen en tres grupos, a saber: clases que modelan componentes de alto nivel, clases que modelan componentes de bajo nivel y clases que modelan conectores. La jerarquía de conceptos *ArchitecturalComponent* representa componentes de alto nivel, por lo que incluye componentes de aplicación (*ApplicationComponent*) y componentes de administración (*ManagementComponent*). Por su parte, los conceptos modelados a partir de *FunctionalComponent* definen los componentes de bajo nivel a ser usados para definir el comportamiento de componentes de alto nivel. A su vez, las clases *ExternalLink* e *InternalLink* (junto con sus especializaciones) definen el conjunto de relaciones disponibles para modelar vínculos entre estos componentes.

Un *ArchitecturalComponent* posee una entrada (*Input*) y una salida (*Output*) sobre las cuales se indican sus conexiones, las cuales se definen utilizando las especializaciones de *ExternalLink* (definidas como *UnidirectionalExternalLink*, *BidirectionalExternalLink* y *SpecialUnidirectionalExternalLink*). Las especializaciones de *ManagementComponent* modelan los elementos que gestionan el desempeño de la aplicación, mientras que las clases que heredan de *DefinedApplicationComponent* modelan componentes de aplicación definidos. En cambio, los componentes de aplicación no definidos deben

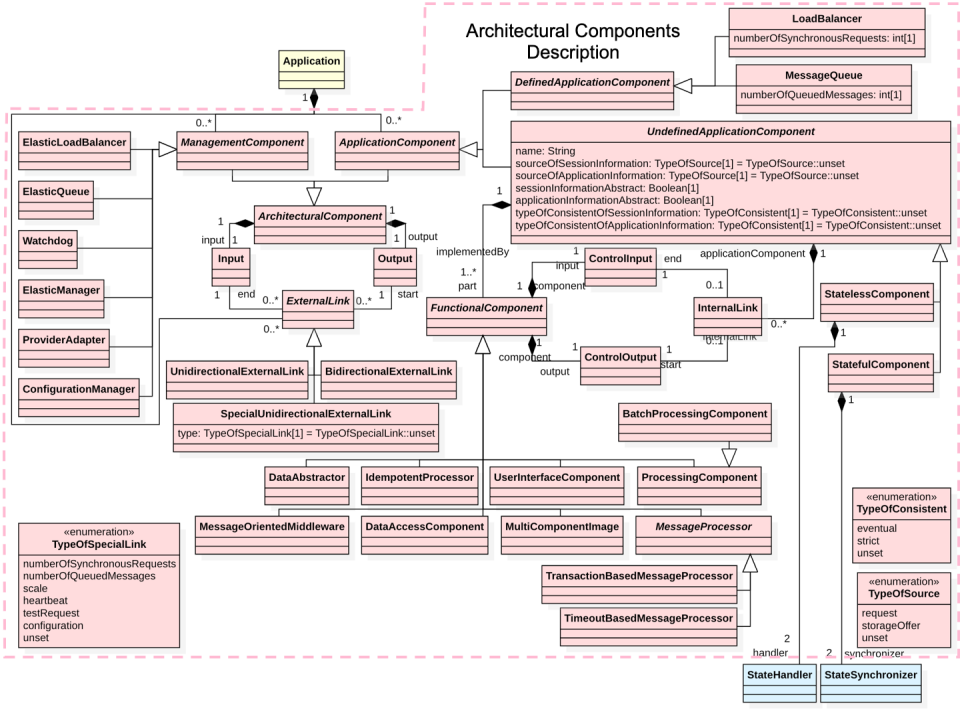

Figura 4 – Metamodelo arquitectónico (parte 3).

crearse en base a las especializaciones de *UndefinedApplicationComponent*. En este contexto, se tiene que un *StatelessComponent* es un componente cuyo estado se maneja de forma externa, mientras que en un *StatefulComponent* todas sus instancias deben sincronizar su estado para proveer un comportamiento uniforme. El comportamiento de *UndefinedApplicationComponent* debe especificarse haciendo uso de múltiples instancias de la clase *FunctionalComponent*. Tales elementos modelan responsabilidades específicas, por lo que su combinación permite generar nuevas funciones de mayor complejidad. Para esto, disponen de una entrada *ControlInput* y una salida *ControlOutput* sobre las cuales se describen las interacciones con sus pares en base a instancias de la clase *InternalLink*.

Finalmente, la sección *components state description* tiene los elementos que deben usarse para especificar el manejo de estado en componentes de aplicación definidos (Figura 5, recuadro celeste). El concepto *StateManager* modela un elemento genérico de control que gestiona estados según su componente asociado (en un *StatelessComponent* se usa un *StateHandler* y en un *StatefulComponent* se usa un *StateSynchronizer*). Cada componente tiene dos estados para manejar: estado de sesión (clase *SessionState*) y estado de aplicación (clase *ApplicationState*), los cuales deben asociarse al tipo de información a manipular (clases *SessionInformation* y *ApplicationInformation*, respectivamente).

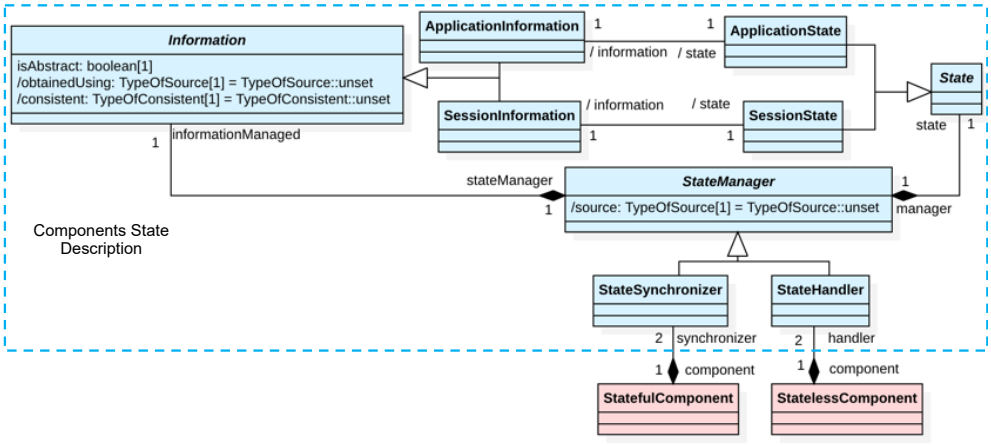

Figura 5 – Metamodelo arquitectónico (parte 4).

A fin de complementar la descripción UML, se incorporaron al modelo restricciones OCL (formuladas como invariantes) que buscan garantizar la correcta instanciación de los elementos arquitectónicos definidos. Por ejemplo, "el inicio y el fin de un *ExternalLink* no pueden referir al mismo *ArchitecturalComponent*" (Figura 6).

```
context ExternalLink
inv: self.start.architecturalComponent<>self.end.architecturalComponent
```

Figura 6 – Ejemplo de invariante OCL complementaria a la descripción UML.

4.3. Instanciación de Arquitectura: Herramienta de Modelado Gráfico

El objetivo de una descripción arquitectónica es brindar un mecanismo de soporte al proceso de diseño. Sin embargo, es deseable que este mecanismo pueda llevarse a un formato computacional que permita y facilite su análisis e instanciación (Albin, 2003).

Teniendo en cuenta que Eclipse es uno de los entornos de desarrollo más usados, en (Blas, Gonnet & Leone, 2015) se propone un plug-in para este entorno que facilita la instanciación gráfica del metamodelo. En su implementación se usaron dos herramientas del proyecto Modeling: EMF y GMF. El diagrama UML y las restricciones OCL que lo complementan se definieron en un modelo Ecore (especificado con EMF), mientras que las descripciones gráficas asociadas a los componentes y vínculos que conforman el metamodelo se detallaron con GMF.

La Figura 7 presenta la pantalla principal del plug-in junto con el modelado de la arquitectura de tres bandas propuesta en (Fehling, Leymann, Retter, Schupeck & Arbitter, 2014). Como puede observarse, existen cuatro sectores: *explorador de proyectos* (área 1, indica el proyecto en el que se diseña la arquitectura), *área de trabajo* (área 2, espacio para generar el diseño), *paleta de herramientas* (área 3, elementos disponibles para el modelado) y *tabla de propiedades* (área 4, atributos de los elementos instanciados). Luego, en relación a un proyecto de software (parte 1), el arquitecto construye su diseño web (en la parte 2) utilizando los elementos disponibles (de la parte 3) y configurando sus propiedades (parte 4). Además de las funcionalidades de modelado, la herramienta incluye un asistente de creación que ayuda al arquitecto en la especificación del entorno de CC sobre el cual se despliega la aplicación diseñada.

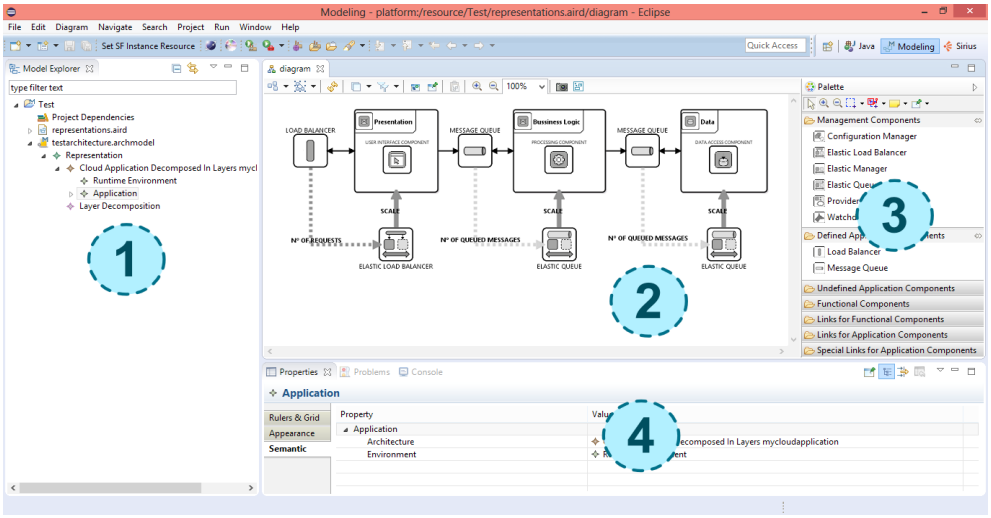

Figura 7 – Plug-in de Eclipse implementado para instanciar gráficamente el metamodelo.

5. Evaluación de Instancia: Verificación de Patrones de Diseño

Aun cuando la herramienta permite instanciar arquitecturas de CC válidas en base a los elementos y las reglas impuestas, es deseable incorporar información referida a los patrones de diseño utilizados. Luego, se incorporaron invariantes OCL que ayudan a verificar la consistencia de los diseños instanciados según los patrones arquitectónicos propuestos en el nivel *Cloud Application* (Tabla 1).

Esta verificación se realiza automáticamente mediante una opción del menú que comprueba las invariantes OCL sobre el conjunto de instancias creadas por el arquitecto. Ante una violación de restricciones, se informa al arquitecto indicando los problemas detectados sobre los componentes definidos. Por el contrario, si no se detectan inconvenientes se comunica al arquitecto que el diseño es correcto.

Luego, por medio de la verificación de los patrones de diseño, se provee un primer mecanismo de evaluación de arquitecturas de CC en términos de las características conocidas de los esquemas de diseño aplicados.

6. Conclusiones y Trabajos Futuros

Teniendo en cuenta que el paradigma de CC se ha convertido rápidamente en una de las estrategias de solución tecnológica más populares e influyentes del mundo actual, en este trabajo se han especificado los lineamientos requeridos para la construcción de un entorno de trabajo que facilite la tarea de diseño arquitectónico. La base de dicho entorno queda definida como un metamodelo, el cual describe un conjunto de elementos obtenidos a partir del estudio de múltiples patrones de diseño. Sobre este metamodelo, se han implementado los módulos de instanciación y verificación de patrones de diseño, tendientes a componer el entorno final. Luego, al restringir los componentes disponibles, el arquitecto debe tomar una menor cantidad de decisiones que, posteriormente, pueden ser verificadas conforme patrones de diseño de CC preestablecidos.

Para la especificación del módulo *decisiones arquitectónicas*, se tomará como base el modelo de gestión de conocimiento propuesto en (Roldán, Gonnet & Leone, 2016) que captura las operaciones ejecutadas junto con los elementos arquitectónicos que se operaron, los resultados obtenidos y los objetivos a cumplir. De esta manera, se complementará la propuesta desarrollada en este trabajo con un nuevo módulo.

Referencias

Albin, S. T. (2003). *The art of software architecture: design methods and techniques*. Hoboken: John Wiley & Sons.

Atkinson, C., & Kuhne, T. (2003). Model-driven development: a metamodeling foundation. *IEEE software*, 20(5), 36–41.

Barbacci, M., Klein, M., Longstaff, T., & Weinstock, C. (1995). Quality Attributes (CMU/SEI-95-TR-021). Pittsburgh*: SEI, Carnegie Mellon University*.

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

Blas, M. J., Gonnet, S., & Leone, H (2015). Un Modelo para la Representación de Arquitecturas Cloud basadas en Capas por medio de la Utilización de Patrones de Diseño: Especificación de la Capa de Aplicación. In *Proceedings of 3º Congreso Nacional de Informática / Sistemas de Información (CONAIISI),* Buenos Aires.

Carignano, M. C., Gonnet, S., & Leone, H. (2019). KE-SER: Un sistema basado en el conocimiento y la experiencia para dar soporte a arquitectos de software en aspectos de seguridad. *RISTI Revista Ibérica de Sistemas e Tecnologias de Informação*, 32, 97–112.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., & Arbitter, P. (2014). *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Berlin: Springer Science & Business Media.

Garlan, D., & Shaw, M. (1994). An introduction to software architecture (CMU-CS-94-166). Pittsburgh*: Carnegie Mellon University*.

Jansen, A., & Bosch, J. (2005). Software architecture as a set of architectural design decisions. In Procedings of *5th Working IEEE/IFIP Conference on Software Architecture* (pp. 109-120).

Kazman, R., Gagliardi, M., & Wood, W. (2012). Scaling up software architecture analysis. *Journal of Systems and Software*, 85(7), 1511–1519.

Kitchenham, B., & Pfleeger, S. L. (1996). Software quality: the elusive target. *IEEE software*, 13(1), 12–21.

Kruchten, P., Capilla, R., & Dueñas, J. C. (2009). The decision view's role in software architecture practice. *IEEE software*, 26(2), 36–42.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing (SP 800-145). *NIST*. Doi: 10.6028/NIST.SP.800-145.

Pahl, C., Giesecke, S., & Hasselbring, W. (2009). Ontology-based modelling of architectural styles. *Information and Software Technology*, 51(12), 1739–1749.

Palos-Sánchez, P., Arenas-Márquez, F. & Aguayo-Camacho, M. (2017). La adopción de la tecnología cloud computing (SaaS): efectos de la complejidad tecnológica vs formación y soporte. *RISTI Revista Ibérica de Sistemas e Tecnologias de Informação*, 22, 89–105.

Pressman, R. (2010). *Software engineering: a practitioner's approach*. New York: McGraw-Hill.

Roldán, M., Gonnet, S., & Leone, H. (2016). Operation-based approach for documenting software architecture knowledge. *Expert Systems*, 33, 313–348.

Silva, A., Verdúm, J., Espinoza, M., Hurtado, D., & Poma, A. (2015). Modelo de calidad de servicio QoS en entornos cloud. *International Journal of Information Systems and Software Engineering Big Companies*, 2(2), 70–80.

Sommerville, I. (2005). *Ingeniería del software*. London: Pearson Educación.

Tyree, J., & Akerman, A. (2005). Architecture decisions: Demystifying architecture. *IEEE software*, 22(2), 19–27.