Un lenguaje de modelado para representar visualmente las decisiones de diseño arquitectónico y su *rationale*

A modeling language for visually representing architectural design decisions and their *rationale*

Milton Sánchez-Grueso[1]
Julio Hurtado-Alegría[2]

[1]Universidad del Cauca (Colombia). Correo electrónico: sanchezg@unicauca.edu.co
orcid: https://orcid.org/0000-0003-2109-4969

[2]Universidad del Cauca (Colombia). Correo electrónico: ahurtado@unicauca.edu.co
orcid: http://orcid.org/0000-0002-2508-0962

Recibido: 14-01-2020 Aceptado: 30-07-2020

Cómo citar: Sánchez-Grueso, Milton; Hurtado-Alegría, Julio (2020). Un lenguaje de modelado para representar visualmente las decisiones de diseño arquitectónico y su *rationale*. *Informador Técnico*, *84*(2), 155-174.
https://doi.org/10.23850/22565035.2622

Resumen

El *rationale* arquitectónico es el conjunto de razones detrás de las decisiones tomadas al diseñar la arquitectura de un sistema de software. Normalmente, dicho *rationale* se queda en las mentes de los diseñadores y demás involucrados en el diseño. Por lo tanto, el razonamiento detrás de las decisiones que sustentan el modelo de arquitectura puede perderse si no se documenta adecuadamente, causando problemas de mantenibilidad en el software. En la práctica, el *rationale* no se documenta o se documenta en medio de las descripciones arquitecturales, lo cual dificulta su comprensión y apoyo a las decisiones posteriores, dentro del desarrollo y mantenimiento de software, lo cual resulta más crítico en el enfoque ágil de desarrollo. Para abordar este problema, en este trabajo se propone un enfoque de documentación que combina el modelado del *rationale*, con foco en las decisiones que se toman en proyectos que utilizan métodos ágiles, con el fin de especificar un lenguaje que fundamente las bases para la construcción de una herramienta que hemos denominado *Decisions and rationale modeling language* (DRML). El lenguaje es evaluado para documentar el *rationale* en el marco del proyecto "Sistema Único de Información Indígena (SUIIN)", en el contexto de una entidad pública que dentro sus procesos tienen un equipo de trabajo conformado por ingenieros de sistemas, enfocados en el desarrollo de software. Dicha evaluación ha permitido establecer que el enfoque brinda la suficiente expresividad para documentar las decisiones y su *rationale*, sin embargo, presenta limitaciones para escalar el modelado para un número grande de decisiones y sus relaciones.

Palabras clave: *rationale*; arquitectura de software; decisiones de diseño arquitectónico; modelo de decisiones; *rationale* arquitectónico.

Abstract

The architectural *rationale* is the set of reasons behind the decisions made when designing the architecture of a software system. Normally, this *rationale* remains in the minds of designers and others involved in the design. Therefore, the reasoning behind the decisions that underpin the architecture model may be lost if not properly documented, causing maintainability problems in the software. In practice, the *rationale* is not documented or is documented in the middle of the architectural descriptions, which makes it difficult to understand and

support subsequent decisions, within the development and maintenance of software, which is more critical in the agile approach to development. To address this problem, this paper proposes a documentation approach that combines *rationale* modeling with a focus on decisions made in projects that use agile methods, to specify a language that provides the basis for the construction of a tool that we have called DRML (Decisions and *Rationale* Modeling Language). The language is evaluated to document the *rationale* in the framework of the Unique Indigenous Information System (SUIIN, for its acronym in Spanish) project, in the context of a public entity that within its processes has a work team made up of systems engineers focused on software development. This evaluation has established that the approach provides sufficient expressiveness to document the decisions and their *rationale*, however, it has limitations to scale the modeling for a large number of decisions and their relationships.

Keywords: *rationale*; software architecture; architectural design decisions; decision model; architectural *rationale*.

1. Introducción

La arquitectura de software ha sido identificada como un artefacto de gran valor para la evolución de los sistemas de software. Sin embargo, es difícil que las organizaciones de software le encuentren utilidad al documento de arquitectura, porque no lo encuentran lo suficientemente completo, es muy extenso o resulta incomprensible para los diferentes interesados. Por lo anterior, hay una gran distancia entre el valor potencial y el valor real de las arquitecturas de software para las organizaciones. En proyectos ágiles, cambia la forma de diseñar la arquitectura, debido a que no es una actividad de una sola persona, las decisiones de diseño se toman en grupo y se centran en la entrega continua del producto y en poco tiempo. A pesar de esto, los equipos hacen un esfuerzo en combinar prácticas de arquitectura con agilidad, con el fin de obtener productos con calidad y en muy poco tiempo (Lopes; Aquino, 2017). Además, existen problemas durante el diseño de la arquitectura, relacionados con la toma temprana de decisiones de diseño, con un *rationale* limitado y con sesgos cognitivos acerca del problema que se está tratando de resolver, así como el impacto de estas primeras decisiones sobre el resto del desarrollo (van Vliet; Tang, 2016). Alrededor de la arquitectura de software existen mitos y creencias, sustentados por casos de éxito o fracaso, los cuales van desde la cantidad de esfuerzo necesario para la documentación, hasta el tamaño del equipo o las personas responsables de tomar decisiones de diseño arquitectural. La mayoría de las creencias se basan en la idea de que el resultado del proyecto depende en gran medida de los métodos utilizados durante el diseño y la toma de decisiones (van Der Ven; Bosch, 2016). Durante la creación de una arquitectura de software, los arquitectos y los demás interesados toman decisiones. Estas decisiones pueden estar directamente relacionadas con los requisitos funcionales o de calidad. Algunas decisiones de diseño se toman sobre la marcha, de manera más o menos arbitraria, de acuerdo a la experiencia personal, el conocimiento del dominio, las restricciones financieras y la experiencia disponible, entre otros aspectos. Las decisiones, así como las razones de esas decisiones, a menudo no son explícitas por adelantado, son implícitas y, por lo general, permanecen indocumentadas (Roeller; Lago; van Vliet, 2006). Facilitar la trazabilidad bidireccional entre los aspectos de calidad, las decisiones arquitectónicas, el *rationale* y los módulos afectados, incluyendo una perspectiva de código, brinda el apoyo crítico para varias áreas del proceso de ingeniería de software. Una lista incompleta de áreas es el análisis de impacto de cambios, validación de requisitos, preservación arquitectónica, construcción de casos de seguridad y, a largo plazo, el mantenimiento del sistema. Por ejemplo, la práctica ha demostrado que la erosión de la arquitectura ocurre a menudo cuando los desarrolladores realizan cambios en el código, sin comprender completamente las decisiones arquitectónicas subyacentes y las preocupaciones relacionadas con la calidad (Cleland-Huang; Mirakhorli; Czauderna; Wieloch, 2013). Hoy este hecho desafortunadamente es una realidad, ya que la arquitectura de software es definida por muchos como la composición de un conjunto de decisiones de diseño arquitectónico, donde lo que se busca es evitar la evaporación del conocimiento de las decisiones de diseño, ya que se han ido convirtiendo en una parte explícita de la arquitectura. Esto a su vez, muestra el alto costo que tiene el cambio en las arquitecturas de software, lo que evidencia la complejidad de estos cambios y cómo ésta se erosiona durante su evolución. Los problemas se deben a la pérdida del conocimiento. Actualmente, la información sobre las decisiones de diseño arquitecturales queda implícitamente ilustrada, pero

es carente de una representación explícita. En consecuencia, el conocimiento sobre estas decisiones de diseño arquitectónico desaparece. Por ejemplo, durante el diseño, desarrollo, evolución, reutilización e interpretación, el *rationale* de las decisiones tomadas queda fuera del alcance para la nueva toma de decisiones. En el diseño, la principal preocupación es cuál decisión tomar. En desarrollo es importante saber qué y por qué se han tomado ciertas decisiones de diseño. La evolución de la arquitectura tiene que ver con las nuevas decisiones de diseño o eliminar las antiguas para satisfacer los requisitos cambiantes. El reto es hacer esto en armonía con las decisiones de diseño existentes, en particular, entender las razones que llevaron a la toma de estas decisiones (Jansen; Bosch, 2005).

El artículo propone un lenguaje integrado para expresar en forma de modelo las decisiones de diseño arquitectónico, así como el *rationale* que las respalda dentro de un contexto ágil con prácticas de arquitectura de software. Para ello, se ha definido una meta-modelo como sintaxis abstracta y un conjunto de representaciones como sintaxis concreta, para que los arquitectos puedan expresar sus decisiones y el *rationale* detrás de ésta. Además, se usó el meta-modelo del lenguaje en la construcción de una herramienta, con un enfoque de ingeniería dirigida por modelos (MDE por sus siglas en inglés) para facilitar al arquitecto el modelado y su almacenamiento. El lenguaje ha sido evaluado en una prueba piloto en una empresa de software del Suroccidente colombiano, haciendo uso de la herramienta de modelado propuesto. El estudio de caso ha permitido evidenciar que la herramienta permite expresar la mayoría de aspectos relacionados con las decisiones y su *rationale*, así como sus limitaciones para escalar los modelos y relacionar las decisiones de diseño.

2. Marco teórico y estudios previos

Kruchten, Obbink y Stafford (2006) definen la arquitectura de software como la estructura y organización de componentes y subsistemas que pueden interactuar entre sí, con el fin de formar sistemas complejos con una esperanza de vida mucho más alta. Las arquitecturas de software se comprenden de componentes, conectores y restricciones representados mediante diferentes perspectivas, denominadas vistas. Estas vistas proveen una descripción consistente y complementaria de la arquitectura de software, las cuales representan las diferentes preocupaciones de los interesados del software, además de servir como un medio de comunicación de la arquitectura y su razón de ser (Roldán; Gonnet; Leone, 2016).

Los diseñadores de arquitectura de software inevitablemente trabajan con patrones de arquitectura y con tácticas. Los patrones de arquitectura describen la estructura y el comportamiento de alto nivel de los sistemas de software como la solución a múltiples requisitos del sistema, mientras que las tácticas son decisiones de diseño que mejoran los problemas de atributos de calidad individuales. Las tácticas que se implementan en las arquitecturas existentes pueden tener un impacto significativo en los patrones de arquitectura en el sistema. De manera similar, las tácticas que se seleccionan durante el diseño inicial de la arquitectura tienen un impacto significativo en la arquitectura del sistema que se diseñará, especificando qué patrones usar y cómo se deben cambiar para acomodar las tácticas (Harrison; Avgeriou, 2010).

Los atributos de calidad son características que tiene el sistema, como usabilidad, mantenibilidad, rendimiento y confiabilidad. Normalmente, los sistemas tienen múltiples atributos de calidad importantes, y las decisiones tomadas para satisfacer un atributo de calidad particular pueden afectar a otro atributo de calidad. Asimismo, las tácticas son medidas tomadas para mejorar los atributos de calidad. Las tácticas impactan los patrones de arquitectura de varias maneras. En algunos casos, una táctica puede implementarse fácilmente utilizando las mismas estructuras (y comportamiento compatible), como un patrón de arquitectura particular. Por otro lado, una táctica puede requerir cambios significativos en la estructura y el comportamiento del patrón o estructuras y comportamientos completamente nuevos. En este caso, la implementación de la táctica y el mantenimiento futuro del sistema son considerablemente más difíciles y propensos a errores. Las tácticas pueden ser "tiempo de diseño" o enfoques generales de diseño e implementación, como "ocultar información" para mejorar la modificabilidad, o "tácticas de tiempo de ejecución", que son características dirigidas a un

aspecto particular de un atributo de calidad, como "autenticar usuarios" para mejorar la seguridad (Harrison; Avgeriou, 2010).

Una decisión de diseño arquitectónico es definida como la descripción del conjunto de adiciones, sustracciones y modificaciones a la arquitectura del software, su lógica, reglas de diseño, restricciones de diseño y los requisitos nuevos que traen consigo nuevas decisiones (Jansen; Bosch, 2005). Por su parte, el *rationale* o fundamento arquitectónico se refiere a la justificación de una decisión de diseño arquitectónico. Reynoso (2004) define el *rationale* como la "base subyacente para la arquitectura en términos de restricciones derivadas de los requerimientos del sistema (p.9)". El *rationale*, como componente de la arquitectura software, se convierte en un punto crítico para diseñarla, describir las decisiones, evolucionarla y tomar nuevas decisiones. Tang, Babar, Gorton y Han (2006) exploran el valor del *rationale* desde el punto de vista del diseño, buscando documentar el conocimiento de fondo con sus decisiones de diseño y reflejar la importancia que tiene para los arquitectos, documentar el *rationale* como alternativa para analizar las decisiones.

Para el diseño y la documentación de la arquitectura software, se hace uso de los lenguajes de definición de arquitecturas (ADL), los cuales proveen elementos para modelar la arquitectura conceptual de un sistema software, distinguiéndola de su implementación. Algunos ADL son genéricos, por ejemplo, el Lenguaje Unificado de Modelado (UML) y otros específicos al dominio de las aplicaciones. Los más conocidos lenguajes son: acme, aesop y sad (Hernández; Hurtado, 2016). En cuanto a la ingeniería dirigida por modelos (MDE), es un paradigma de desarrollo de software que apunta a elevar el nivel de abstracción, enfocándose en actividades de modelado, en lugar de codificación. Según el paradigma MDE, a partir de un modelo y mediante transformaciones, es posible obtener automáticamente una variedad de artefactos, como nuevos modelos y código, entre otros. En este contexto, el desarrollo de software puede verse como un proceso de transformación, donde los modelos de abstracción de bajo nivel se obtienen automáticamente o semiautomáticamente mediante la transformación de modelos de abstracción de alto nivel (Bucaioni; Cicchetti; Ciccozzi; Mubeen; Sjodin 2017).

Aldrich, Chambers, Notkin (2002) presentan ArchJava, una pequeña extensión Java que integra las especificaciones de arquitectura de software en el código de implementación Java, buscando que la implementación se ajuste a las restricciones arquitectónicas. Para evaluar su enfoque, implementan la extensión a una aplicación de diseño de circuitos, usando un diagrama informal de la arquitectura dibujado a mano, el cual usan como guía para hacer esta arquitectura explícita en el código. Finalmente, concluyen que ArchJava permite a los programadores expresar la estructura arquitectónica para luego ser llenada en la aplicación con código Java. Este trabajo muestra la importancia de poner la arquitectura de manifiesto en el código, pero usa otro enfoque, el de representar las abstracciones. Además, no tiene en cuenta el *rationale* y las decisiones de diseño arquitectónico que no se pueden representar en el código.

Cleland-Huang *et al*. (2013) presentan un enfoque centrado en la arquitectura para hacer seguimiento a las preocupaciones de calidad de las partes interesadas, tales como fiabilidad, disponibilidad, seguridad, integridad, rendimiento, portabilidad, los requisitos arquitectónicamente significativos, las razones de diseño y el código fuente. En la trazabilidad centrada en decisiones (DCT por su sigla en inglés), todos los vínculos de rastreo se centran en decisiones arquitectónicas que incluyen factores tan variados, como plataformas, lenguajes, frameworks, patrones de diseño de alto nivel, mecanismos de comunicación, tácticas arquitectónicas de bajo nivel y estilos arquitectónicos, entre otros. Hoy los sistemas de software están diseñados para satisfacer los requisitos funcionales, así como una amplia escala de intereses de calidad relacionados con los atributos antes mencionados (Bass Clements; Kazman 2003). Por ejemplo, un arquitecto puede decidir abordar una meta de portabilidad mediante el uso de un estricto enfoque en capas, que simplifica el proceso de creación de nuevas interfaces graficas de usuario (GUIs) específicas de la plataforma, o para lograr una meta de disponibilidad mediante la utilización de la táctica de control para supervisar el estado de salud de un componente crítico, o cumplir una meta de rendimiento a través de la utilización de un conjunto de hilos para administrar recursos compartidos (Cleland-Huang *et al*., 2013). Facilitar la trazabilidad bidireccional entre las preocupaciones de calidad, las decisiones arquitectónicas, las justificaciones y las áreas pertinentes del código, brinda apoyo crítico

a varias disciplinas del proceso de ingeniería de software (Cleland-Huang *et al*., 2013). Por ejemplo, la práctica ha demostrado que la erosión de la arquitectura ocurre a menudo, cuando los desarrolladores realizan cambios en el código sin comprender completamente las decisiones arquitectónicas subyacentes y sus preocupaciones relacionadas con la calidad (Cleland-Huang *et al*., 2013). Los enlaces de rastreo que se utilizan para comunicar las principales decisiones de diseño a nivel arquitectónico van en línea con el enfoque establecido por este trabajo, donde se busca modelar el *rationale* y las decisiones de diseño en un contexto ágil.

Hadar, Sherman, Hadar y Harrison (2013) presentan un estudio de caso que tiene como objetivo, identificar las dificultades que encuentran los arquitectos y otras partes interesadas, cuando documentan la arquitectura en el desarrollo ágil. Los hallazgos muestran que el documento que contiene la especificación de la arquitectura suele ser muy extenso, complejo y, en muchos casos, no es capaz de explicarse así mismo. Con el fin de ajustar la documentación de la arquitectura al enfoque de documentación ligera y mínima de los procesos ágiles, estos autores proponen un documento de especificación más corto, que requiera esfuerzos reducidos de documentación, el cual resulta en una documentación simplificada que es más fácil revisar, actualizar y comunicar. Los documentos de arquitectura, por lo general, suelen ser extensos y complejos, y pueden crecer de decenas a cientos de páginas; que consisten en múltiples documentos que dentro y entre ellos abarcan múltiples conceptos, relaciones, vistas y niveles de abstracción. Jansen, Avgeriou y van Der Ven (2009) identifican una lista de desafíos relacionados con la documentación de arquitectura que se basa en los siguientes tres desafíos: primero, comprensibilidad de los documentos por parte de los arquitectos; segundo, localización del conocimiento relevante de la arquitectura y, tercero, mantener la documentación de la arquitectura actualizada.

Hesse, Kuehlwein, Paech Roehm y Bruegge (2015) hablan de la importancia que tiene para un equipo de desarrollo de software documentar las decisiones de implementación. Cuando se revisa el código durante el mantenimiento, las decisiones detrás de la arquitectura deben entenderse y posiblemente ajustarse a la situación actual. Por lo tanto, el conocimiento de la decisión raramente se documenta volviéndose inaccesible, especialmente, cuando los desarrolladores ya no hacen parte del equipo. Esto obstaculiza el mantenimiento eficaz. Para solucionarlo, los autores han desarrollado un modelo de anotación para el conocimiento y la decisión integrados a través de una herramienta de gestión del conocimiento llamada UNICASE, una extensión de Eclipse que proporciona un modelo para documentar el conocimiento de las decisiones previamente tomadas, pero a nivel de diseño. El enfoque permite a los desarrolladores documentar decisiones dentro del código. En este proyecto se busca de manera similar, documentar las decisiones de diseño arquitectónico a través de la incorporación de un complemento software (plug in) en el ambiente de desarrollo eclipse a nivel de modelado.

Para la estructuración del lenguaje, se han tomado únicamente los elementos de documentación que se consideraron relevantes para el propósito del modelado que involucra las decisiones y su *rationale* para ser aplicado al contexto ágil. Por ello, esta propuesta toma como punto de partida el meta-modelo planteado por Plataniotis, Ma, Proper y de Kinderen (2015). Su principal contribución es un meta-modelo formal que captura el *rationale* y las interrelaciones de las decisiones de diseño; el trabajo de Dorado y Hurtado (2019), donde documentan el *rationale* mediante anotaciones en el código; y la definición realizada en van Der Ven Jansen, Nijhuis y Bosch (2006), quienes proponen integrar el *rationale* y los artefactos arquitectónicos en el concepto de una decisión de diseño, que a su vez combina el *rationale* de la arquitectura de software. El enfoque propuesto por Gilson y Englebert (2011), que considera el modelado de requisitos arquitectónicamente significativos y el modelado de arquitectura, en donde restricciones y requisitos se adjuntan a las construcciones arquitectónicas, y cualquier modificación en el modelo de arquitectura resultante de una decisión tomada desde el modelo de requisitos, se registra como una transformación del modelo. La metodología de Che (2014), que documenta explícitamente las decisiones de diseño arquitectónico, utilizando un enfoque basado en escenarios, abarca una serie de vistas de arquitectura de software para registrar el conocimiento arquitectónico, mediante características centradas en gestionar la evolución de las decisiones de diseño arquitectónico, con el fin de reducir la evaporación del conocimiento asociado. La plantilla propuesta por Dermeval *et al*. (2013), que captura el *rationale* de las decisiones de diseño arquitectónico, relacionando los requisitos funcionales, requisitos de calidad, interesados y *rationale*. Igualmente, el modelo propuesto se basa en el modelo de decisiones planteado

por Manteuffel, Tofan, Koziolek, Goldschmidt, Avgeriou (2014), que sugiere una herramienta soportada en el meta-modelo de la ISO/IEC/IEEE 42010 para documentar decisiones y su conocimiento implícito. Finalmente, es importante tener en cuenta que la toma de decisiones de arquitectura de software no es una actividad individual, sino un proceso grupal, donde las decisiones de diseño arquitectónico son tomadas por grupos de partes interesadas, heterogéneas y dispersas (Malavolta; Muccini; Rekha, 2014).

Se realizó un análisis de los estudios para determinar cuáles de estos tenían mayor relevancia para la investigación en curso, es decir, que brinden elementos conceptuales y teóricos para construir el lenguaje de documentación de *rationale*. La Tabla 1 muestra los estudios que influyeron en la construcción del lenguaje y sus abstracciones más importantes.

Tabla 1.
Trabajos relacionados con el rationale y las decisiones de diseño arquitectónico

Agrupaciones lenguaje DRML	*Rationale*	Contexto			Consecuencia				Alternativa	Decisión arquitectónica		Justificación			Táctica	Patrón	Estrategia
Elementos por el estudio	R	M	P	C	To	Cn	Pr	Cq	Alt	ADD	D	Qa	BR	QG	T	P	E
(Cleland-Huang et al., 2013)	X									X				X			
(van Der Ven et al., 2006)		X	X	X	X	X	X	X									
(Dermeval et al., 2013)								X	X		X						
(Dorado y Hurtado, 2019)	X	X	X		X	X	X				X	X	X	X	X	X	
(Gilson y Englebert, 2011)									X		X						
(Plataniotis et al., 2015)									X		X						
(Harrison; Avgeriou, 2010)											X	X		X	X	X	
(Jansen; Bosch, 2005)		X	X	X	X					X	X						
(Che, 2014)	X				X				X							X	
Lenguaje DRML	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Fuente: elaboración propia.

En la Tabla 1 se muestra una comparativa de las agrupaciones que propone el lenguaje *decisions and rationale modeling language* (DRML), con respecto a los elementos de documentación que plantea cada uno de los estudios. En la primera fila se observan las principales abstracciones que el lenguaje DRML agrupa mediante constructos explícitos; la segunda fila muestra los elementos de cada una de las propuestas, en ese orden, se comparan y agrupan: *rationale* (R), motivation (M), problem (P), cause (C); trade-off (To), con (Cn), pro (Pr), consequence (Cq); alternative (Alt), architectural design decisions (ADD), decision (D); quality attribute (Qa), business rule (BR), quality goal (QG); tactic (T), architectural pattern (P) y strategy (E). En el lenguaje DRML propuesto algunos elementos son ciudadanos de primera clase, es decir, existen constructos para modelarlos y

los otros conceptos quedan englobados en ellos, aunque no sean explícitos, sin embargo, cuenta con un espacio general que ofrece la posibilidad de describirlos, por ejemplo, los pros y los contras se relatan libremente en un elemento de tipo consecuencia (Cq).

En la revisión de la literatura se identifican distintos tipos de enfoques que buscan documentar el *rationale* de las decisiones de diseño arquitectónico, por ejemplo, se encuentran herramientas, plantillas, anotaciones en el código, modelos, metodologías, plantillas, encuestas y ciclos de captura, muchos basados en meta-modelos, sin embargo, estas propuestas generan modelos de decisiones, es decir, se centran en generar elementos para documentar la decisión, generando distancia entre el *rationale* y la decisión. Por otra parte, las propuestas pueden llegar a generar una carga cognitiva excesiva al diseñador, esto debido a que incluyen demasiados elementos de documentación. Por ello, es necesario tener un balance, ya que puede ser difícil de mantener una documentación actualizada en un contexto ágil. El enfoque de documentación del *rationale* con anotaciones en el código presenta un gran valor, dado que el código es la fuente más confiable de la información a la hora de hacer mantenimiento del software. Sin embargo, no todas las decisiones y su *rationale* se pueden localizar en un solo lugar del código, puesto que decisiones arquitecturales y su *rationale* pueden impactar muchas partes del sistema, por lo que un modelo que permita documentar en forma explícita y transversal este conocimiento, brindaría una vista más del diseño arquitectónico del sistema, que puede resultar más práctico para el mantenimiento, incluso que extensos documentos de arquitectura y diseño que se tornan poco confiables en el tiempo (Singer, 1998). Así que nuestro enfoque con las anotaciones de código, son dos estrategias que pueden integrarse a través del MDE, de tal forma que brinden una estrategia práctica y simplificada, particularmente, para soportar en lo ágil, con foco en la arquitectura. El primer paso sería extender las anotaciones de código para que los lenguajes sean equivalentes semánticamente para facilitar transformaciones MDE y modelo a texto/código (Model-To-Text o M2T). Así mismo, el lenguaje ofrece los constructos mínimos para integrar las decisiones con el *rationale*, esto permite modelar una vista integradora que es muy necesaria en el modelado de la arquitectura. Por ejemplo, Kruchten, Capilla y Dueñas (2009) en su enfoque de 4+1 vistas hablan de una quinta vista (+1), son los escenarios que permiten evaluar la consistencia entre las vistas. Este modelo cumple este rol, de hacer explícitas las decisiones de diseño que son verificables en las demás vistas y el *rationale* detrás de estas decisiones. De este modo, este trabajo contribuye al área de documentación de la arquitectura, particularmente al enfoque en el cual la arquitectura de software es tratada como un conjunto de decisiones de diseño arquitectónico (Singer, 1998). Es decir, el trabajo de modelar las decisiones y su *rationale*, es parte del modelado arquitectónico que se ocupa de la representación, captura, gestión y documentación de las decisiones de diseño tomadas en el ciclo de vida del software (Kruchten *et al*., 2009).

Diseñar una arquitectura de software es un proceso de toma de decisiones (Lopes y Aquino, 2017). Los métodos ágiles cambiaron drásticamente la forma de diseñar una arquitectura de software.En proyectos que utilizan métodos ágiles, por ejemplo, Scrum, tomar decisiones de diseño arquitectónico no es responsabilidad de una sola persona, sino de todo el equipo de desarrollo, sumado a esto se centran en la entrega continua del producto, sin embargo, vemos como hacen un esfuerzo en incorporar prácticas de arquitectura en sus procesos (Lopes y Aquino, 2017). Nuestro enfoque se ajusta de manera apropiada en este contexto, ya que permite capturar las decisiones y su *rationale* de manera ágil.

3. Decisions and *rationale* modeling language – DRML

En este artículo se presenta un lenguaje de documentación que combina enlaces explícitos entre las decisiones de diseño arquitectónico y sus justificaciones. El lenguaje busca ofrecer un mecanismo para documentar la razón, sus alternativas, justificaciones, consecuencias y contexto detrás de cada decisión de diseño arquitectónico. La técnica propuesta está diseñada para ayudar a las partes interesadas y los arquitectos a compartir, comprender y mantener los diseños de arquitectura. Previamente, se intentó especificar las decisiones de diseño en el código, sin embargo, el código es una sola perspectiva del software, de muchas posibles. Una decisión de diseño que involucre una cualidad de tiempo de ejecución, como es el desempeño, ¿En qué parte del código se

documenta?, considerando que el desempeño es afectado por la forma en que se instalan los componentes, si estos componentes tienen constante comunicación, si van a estar instalados en una misma máquina o servidor, pero, si están en servidores geográficamente dispersos, puede haber ruido, debe hacerse replicación, envío de información, paquetes, aspectos que hacen lento el proceso y que no pueden localizarse en el código. Entonces el código tiene ciertas limitaciones. En lo ágil, uno de los criterios que se aplican en estos casos es que lo que se pueda hacer en código se haga con las anotaciones. El problema con la arquitectura de software, es que no todo se puede especificar a ese nivel, por lo tanto, se necesita una especificación simple, fácil de cargar y adaptable que se pueda agregar por separado como un artefacto más de arquitectura y se pueda asociar con la documentación nueva o existente. Aquí es donde entra nuestro lenguaje, al tratarse de un lenguaje que se puede utilizar a nivel conceptual y de anotaciones, nos basamos en los estudios previos de Dorado y Hurtado (2019), donde se ha dicho que las anotaciones de código tienen un impacto positivo sobre la eficiencia, efectividad y comprensión de la arquitectura de software y su *rationale*, en la realización de cambios arquitectónicos. Por lo tanto, basados en este trabajo, se complementó el modelo de anotaciones con nuestro lenguaje. Se usa el lenguaje para que se puedan representar aquellas anotaciones que no se limiten a solo las posibles anotaciones de código.

La diferencia entre documentar las justificaciones y la decisión muchas veces no es clara. No se pretende proporcionar una frontera entre ellos. De hecho, esta línea depende del dominio de la aplicación, los requisitos de calidad de la arquitectura y la voluntad del arquitecto del sistema. Es común que los arquitectos de software y las partes interesadas sigan estándares de calidad para documentar el diseño de la arquitectura. En ocasiones, de acuerdo a la experiencia de los arquitectos, pueden documentar o no la razón detrás de cada decisión de diseño arquitectónico (Jansen; Bosch, 2005). Es por esto que para documentar las razones de diseño detrás de cada decisión, se optó solo por las que sean significativas de tipo arquitectónico. En la Figura 1, se presenta el meta-modelo que determina el lenguaje de especificación de decisiones de diseño y su *rationale*.

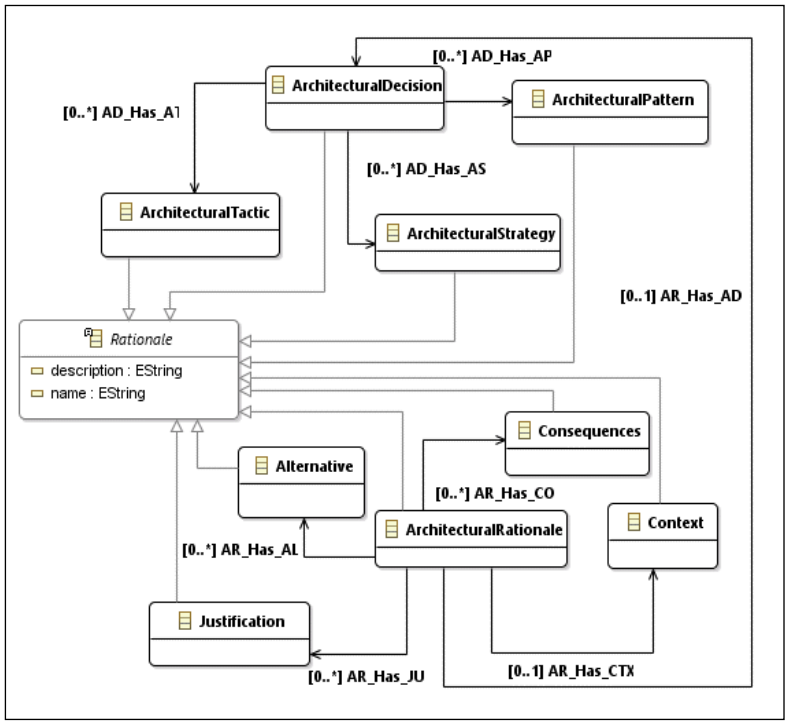

Figura 1. Meta-modelo de Rationale Arquitectónico
Fuente: elaboración propia.

El meta-modelo se compone de las siguientes abstracciones; ArchitecturalRational y ArchitecturalDecision, que se relacionan para proporcionar enlaces concretos entre las razones y la decisión de diseño final. Los enlaces de

documentación permiten a los diseñadores estructurar de manera explícita las razones detrás de una solución. En nuestro enfoque, lo primero que se registra es el *rationale*, mediante el elemento clave *ArchitecturalRationale*, que a su vez se relaciona con *Justification*, que se encarga de agrupar las justificaciones. *Alternative* define un conjunto de alternativas, donde el diseñador debe anotar las alternativas antes de seleccionar una opción en particular. Consequences es una descripción de las consecuencias esperadas a una solución en particular dentro de la arquitectura de software.

El elemento debe proporcionar información adicional a través de los pros y los contras de la solución. Por ejemplo, una toma de decisiones puede introducir la necesidad de tomar otras decisiones, crear nuevos requisitos o nuevas restricciones en el entorno y modificar los requisitos existentes, entre otros. *Context*, la toma de decisiones a menudo está sujeta a diversos factores, por ejemplo, la experiencia del diseñador, el contexto que lo rodea, la tecnología de moda, el curso que acabo de tomar, entre muchos más. Es importante anotar ese conocimiento. *ArchitecturalDecision* representa la decisión de diseño final seleccionada, que a su vez se relaciona con *ArchitecturalPattern*, con el fin de brindar enlaces de seguimiento concretos entre la decisión final y los patrones arquitectónicos relacionados, donde el diseñador puede especificar múltiples soluciones que pueden ser reutilizables en un futuro. *ArchitecturalTactic* es importante documentar el conocimiento acerca de las tácticas, ya que son decisiones de diseño que influyen en la respuesta a los atributos de calidad, en el meta-modelo se relaciona directamente con *ArchitecturalDecision*. Finalmente, *ArchitecturalStrategy*, es vital anotar las estrategias de arquitectura que se componen de un conjunto de tácticas.

3.1. Fundamentos técnicos

El enfoque planteado hace uso de la ingeniería dirigida por modelos (MDE). Para crear la herramienta de modelado específico de dominio, se utilizó *Eclipse Modeling Framework* (EMF) y *Graphical Modeling Framework* (GMF), ambas proporcionadas por la plataforma de Eclipse. GMF precisa de un meta-modelo y para ello se sirve de EMF, que se trata de un marco de modelado que soporta y genera documentos XMI y XML, los cuales son un estándar para el intercambio de información de metadatos con la especificación del dominio.

El primer paso es especificar en *Ecore* (el meta-modelo de EMF) la sintaxis abstracta del lenguaje a través de un meta-modelo, luego GMF establece una serie de pasos para construir una herramienta de este tipo, tomando como entrada el meta-modelo y la definición de la metáfora gráfica del lenguaje (sintaxis concreta) se genera semiautomáticamente de la herramienta (Montenegro; Gaona; Cueva; San Juan, 2011). A partir de este proceso se obtiene un plug in para Eclipse que contiene la herramienta DRML construida. La apariencia de la herramienta se muestra en la Figura 2.

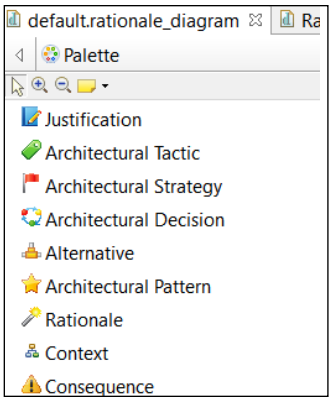

Figura 2. Herramienta para modelar el *Rationale* Arquitectónico
Fuente: elaboración propia.

El funcionamiento de la herramienta es sencillo, para crear los módulos basta con arrastrar los nodos de la "Palete" al área de trabajo, rellenar los campos y conectarlos.

3.2. Características del lenguaje

El lenguaje proporciona varias funciones expresivas que le permiten documentar la razón fundamental, en armonía con la variedad de procesos y su rigurosidad. A continuación, se muestra una descripción general de estas características:

Liviano: el lenguaje con su modelo de dominio puede ser adoptado total o parcialmente por distintos tipos de enfoques, planteando un lenguaje con los elementos necesarios que permita documentar el *rationale*, de las principales decisiones de diseño arquitectónico, pero buscando alivianar el trabajo pesado asociado con las actividades de los arquitectos de software y diseñadores a la hora de documentar y hacer seguimiento.

Simple: plantea los elementos mínimos de trabajo para la documentación del *rationale* arquitectónico, incluyendo y extendiendo los elementos de proceso más comunes encontrados en la literatura científica y los elementos más usados por arquitectos de software y diseñadores.

Expresivo: cada entidad de desarrollo y proyecto es diferente, el lenguaje considera los principales escenarios posibles para documentar un cambio de diseño arquitectónico e incluye varios constructos para armar una gran variedad de decisiones y su *rationale*.

3.3. Enfoque de solución DRML

En este apartado, se presenta la herramienta para la documentación del *rationale* arquitectónico, que se implementó con base en el lenguaje planteado. DRML provee varias características para la documentación, proporciona un editor que soporta la edición específica para el fundamento y la decisión. Primero, dentro de este editor, los arquitectos y diseñadores pueden crear y editar cualquier componente de fundamento arquitectónico en conjunto, como se muestra en la Figura 3, durante la fase de ingeniería de requerimientos, análisis y diseño, implementación, pruebas y despliegue.

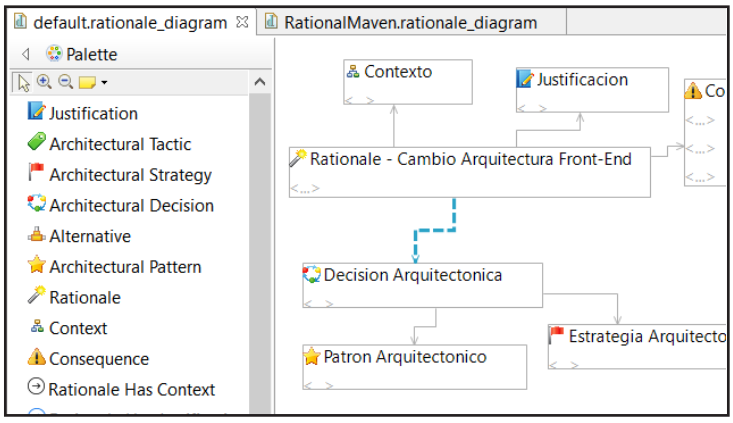

Figura 3. Editor de *Rationale* Arquitectónico
Fuente: elaboración propia.

El lenguaje permite, de una manera sencilla y elegante, crear más de un modelo de fundamento arquitectónico, en donde se puede implementar uno por cada componente de software si así se requiere, al igual que es posible utilizar diferentes elementos donde tan solo con seleccionar y arrastrar del panel de herramientas,

se empieza a construir la documentación. Por ejemplo, utilizar el elemento de *rationale* y *architecturalDecision* y complementarlos con el contexto, justificación, estrategias arquitectónicas, tácticas y las partes que se consideran importantes y que impactan los diseños de la arquitectura de software. Como se muestra en la Figura 4, también es importante resaltar que cada una de las piezas que componen la herramienta son opcionales al momento de construir el conocimiento de la decisión.

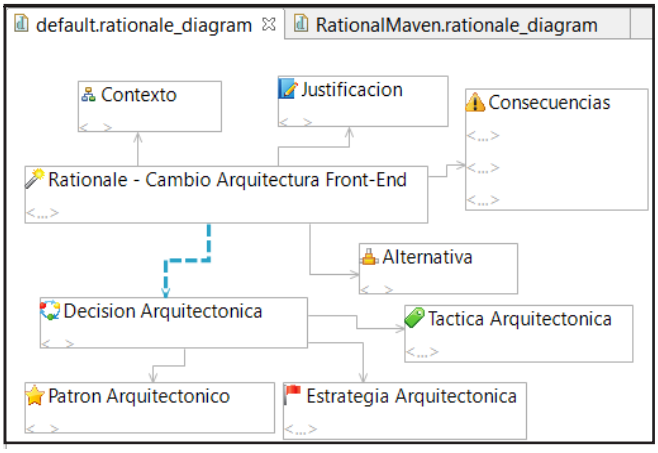

Figura 4. Elementos de *Rationale* Arquitectónico
Fuente: elaboración propia.

El diseño generado con DRML se puede agregar por separado como un artefacto más de arquitectura que se asocia con la documentación nueva y existente. Esto se muestra en la Figura 5.

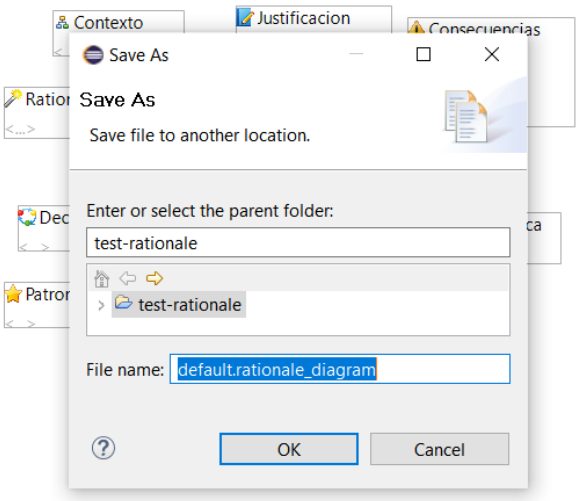

Figura 5. Artefacto de Arquitectura Software
Fuente: elaboración propia.

4. Metodología

Para alcanzar los objetivos propuestos al inicio de este proyecto, se utilizaron las pautas de investigación del método de Bunge (2002) para la especificación del lenguaje de seguimiento que permite facilitar la documentación de arquitecturas. La evaluación del proceso siguió el método de estudios de caso propuesto por Runeson y Höst (2009). El estudio de caso es una metodología de investigación adecuada para la ingeniería del

software, puesto que permite estudiar la ingeniería en su contexto real, buscando mantener la integridad y las características significativas de los eventos, y es ejecutado cuando el investigador tiene poco control sobre los eventos y cuando los sujetos de estudio son más fáciles de observar en grupo, que de manera aislada (Runeson y Höst, 2009). A continuación, se describen las etapas de desarrollo del lenguaje.

Etapa exploratoria: consta del planteamiento del problema, construcción del modelo teórico, reconocimiento de los hechos, estudio del referente teórico de lineamientos y técnicas para documentar el *rationale*, de las principales decisiones de diseño arquitectónico en un contexto ágil con prácticas de arquitectura de software.

Etapa de formulación: se define el mecanismo que permite documentar las principales decisiones de diseño arquitectónico a través de un lenguaje visual de *rationale* arquitectónico.

Etapa de evaluación: se realizan ajustes del proceso siguiendo los hallazgos obtenidos en el estudio de caso y las evaluaciones realizadas.

En las siguientes subsecciones, se utilizan ejemplos de decisiones de diseño del proyecto SUIIN (Sistema Único de Información Indígena) para explicar e ilustrar la herramienta propuesta.

4.1. Contexto del caso

El estudio se realizó en una entidad promotora de salud pública, que busca fortalecer la capacidad administrativa y organizativa de sus procesos y procedimientos internos, a través de la administración automatizada de la información, con el fin de prestar un mejor servicio de salud a sus usuarios y optimizar procesos internos que en algunos casos se hacen manualmente. Razón por la cual, dentro de sus procesos organizativos se implementa un subproceso de desarrollo de software. Se trata de un equipo de trabajo conformado por ingenieros de sistemas, con un alto grado de conocimiento técnico, enfocados en la formalización, optimización y sistematización de procesos, mediante el sistema SUIIN que centraliza la información. Hay que mencionar, además, que implementan buenas prácticas en el desarrollo de sistemas de información. Por ejemplo, la metodología que utilizan para desarrollar software es el proceso unificado ágil (AUP), con el cual se desarrolla software de manera ágil, pero con buenas prácticas de arquitectura. Sin embargo, los sistemas heredados de la organización, que no fueron construidos con estas mismas prácticas, requieren de modificaciones (incluyendo nuevas decisiones de diseño) que requieren comprender y analizar las decisiones previamente tomadas. Por esto, la empresa cuenta con un escenario adecuado para aplicar DRML y su herramienta de soporte. El código fuente del sistema se encuentra en el repositorio de la empresa, con el fin de describir el software con la mayor precisión posible. La estructura se basa en el modelo de vistas de arquitectura “4 +1”.

4.2. Indicador y métrica

La Tabla 2 presenta la métrica utilizada para evaluar el valor que la propuesta tiene sobre la documentación del *rationale* arquitectónico. Posteriormente, se detalla su medición y tratamiento.

Tabla 2.
Indicador y métrica

Indicador	Métrica	Instrumento
Percepción de la usabilidad por parte de los usuarios	Facilidad de uso de los elementos del prototipo implementado a través del meta-modelo DRML.	Encuesta: https://github.com/tomilton/PercepcionUsabilidad.git

Fuente: elaboración propia.

A continuación, se describen en detalle el indicador y la forma en que es calculado a través de la métrica identificada.

4.2.1. Percepción de usabilidad por parte de los usuarios

Para evaluar la satisfacción percibida por los usuarios, se aplicó un cuestionario de siete preguntas basado en el cuestionario estandarizado de Sauro y Lewis (2016). Los ítems del formulario generan cuatro puntajes, uno general y tres subescalas. Las reglas para computar son:

- General: promedio de las respuestas para los ítems 1 al 16 (todos los ítems).
- Calidad de la herramienta: elementos promedio del 1 al 6.
- Calidad de la información: elementos promedio de 7 al 12.
- Calidad de interfaz: elementos promedio del 13 al 16.

La Ecuación 1 es definida para calcular la usabilidad percibida es:

$$Pu = \frac{\sum_{i=1}^{n} RA_i}{n}$$

Donde Pu es la percepción de usabilidad; RAi es el resultado asimilado para la pregunta i, el cual puede tomar valores de 1 al 7; y n es el número de ítems evaluados en la encuesta por los ingenieros.

4.3. Cambios Arquitectónicos

Actualmente el sistema en el front-end está utilizando el *framework* de presentación *primefaces*, el cual ha respondido de manera óptima a las necesidades de la organización, sin embargo, con el fin de mejorar la accesibilidad de los usuarios al sistema, ha surgido la necesidad de que los módulos nuevos que se construyan sean adaptables a dispositivos móviles, además de mejorar su aspecto visual. Por otra parte, el *back-end* de los módulos que se construyen para el sistema no cuentan con un mecanismo que les permita interactuar con otros módulos, esto dificulta la reutilización de código. También, es necesario redefinir algunas capas para lograr una mejor modificabilidad y escalabilidad.

4.4. Participantes

Los participantes fueron ingenieros de software (sistemas y electrónicos) actualmente relacionados con la industria local de software, por lo que las personas seleccionadas tienen conocimiento en los campos de la ciencia de la computación y relacionados. Tres fueron mujeres y tres fueron hombres mayores de 27 años, con experiencia entre 2 y 5 años en la industria, desempeñando diferentes roles, como analista, ingeniero de producto, desarrollador, pruebas y arquitecto de software.

4.5. Desarrollo del Caso

El estudio comienza con una introducción a la teoría del *rationale* arquitectónico para mostrar los conceptos a los participantes, seguido de la presentación de la herramienta DRML. Se realiza un ejemplo para comprender su uso, estructuras, restricciones y el modelado resultante del *rationale*. Después de entrenar a los participantes, se explican los cambios arquitectónicos a realizar y se entrega el material del estudio. La configuración del proyecto se realiza en el entorno de desarrollo Eclipse. El código fuente de la aplicación se descarga del repositorio de control de versiones subversión (SVN por su sigla en inglés) de la empresa, al igual que el documento de arquitectura SAD (documento de arquitectura de software) - SUIIN. También se entrega una guía con los cambios arquitectónicos y una encuesta que deben realizar después de llevar a cabo los cambios arquitectónicos. Los participantes comenzaron a documentar el *rationale* de los cambios, cada participante revisa la documentación previa de un cambio documentado, con el objetivo de evaluar si el *rationale* de cada decisión es comprensible por otro participante. A medida que cada participante modela la decisión con su *rationale* y evalúa la documentación de los demás cambios, se le entrega la encuesta para recopilar información cuantitativa y cualitativa que contribuya a la definición de la estructura final del modelo de documentación para el *rationale* arquitectónico.

5. Resultados

A continuación, se presentan los resultados cuantitativos y cualitativos del estudio de caso.

5.1. Resultados cuantitativos

La Tabla 3 muestra los resultados de la encuesta de percepción de usabilidad para cada uno de los ingenieros en los criterios evaluados: calidad de la herramienta, calidad de la información y calidad de la interfaz.

Tabla 3.
Percepción de usabilidad

Criterios	Unidad de Análisis			
	Ingeniero 1	Ingeniero 2	Ingeniero 3	Promedio
Calidad de la herramienta	6,0	6,6	6,1	6,2
Calidad de la información	5,1	6,6	5,0	5,6
Calidad de interfaz	6,0	6,0	3,2	5,0
Promedio general	5,7	6,4	4,8	5,6

Fuente: elaboración propia.

Los resultados de la Tabla 1 muestran que la calidad de la herramienta es aproximadamente del 89 %. La calidad de la información es 80 % aproximadamente y la calidad de la interfaz es 72 % aproximadamente.

5.2. Resultados cualitativos

Algunas de las observaciones, modelado de *rationale* y decisión de diseño de los ingenieros son presentados a continuación:

a) Observaciones:

La definición de la arquitectura y su lógica son el eje fundamental para la evolución del software, sería interesante contar con un repositorio donde se pudiera buscar el conocimiento documentado. Sería muy útil que la herramienta pudiera subirse a un repositorio y ser instalada como plug in del IDE Eclipse. Sería útil que se pudiera representar especificados patrones de diseño estructural y patrones de miniarquitectura. Sería útil que se pudieran representar los patrones de diseño que hayan sido personalizados. Es necesario que permita enlaces a documentos o artefactos externos, por ejemplo, el lenguaje unificado de modelado (UML). Es importante que en la herramienta se pueda identificar el actor que genera el requerimiento y los diferentes miembros del equipo que participan en el proceso, cada uno con su respectivo rol. Cuando se documentan las tácticas y estrategias arquitectónicas, es útil poder anotar que una estrategia consta de un conjunto de tácticas y relacionarla con la decisión.

b) Modelado de rationale y decisión:

La Figura 6 muestra el modelado final del *rationale* de uno de los ingenieros que participó en el estudio. Se puede ver que hizo énfasis en sustentar la decisión de diseño final; documenta los cambios arquitectónicos, utilizando de manera correcta los elementos que la herramienta le brinda, es decir, se puede ver como el *rationale* se relaciona con tres justificaciones. También anota una posible alternativa dentro de un contexto y una consecuencia con sus pros y contras. Así mismo, documenta la decisión final junto con enlaces explícitos a un patrón específico de arquitectura, que se relaciona con una táctica de arquitectura.

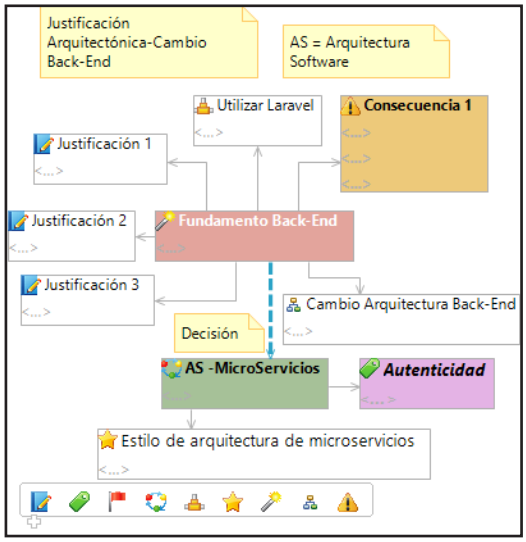

Figura 6. Modelo resultado de *Rationale* Arquitectónico
Fuente: elaboración propia.

La solución propuesta por el ingeniero consiste en cambiar la arquitectura monolítica por una arquitectura basada en microservicios, utilizando el lenguaje Java y Maven (herramienta de software para la gestión y construcción de proyectos Java). Anota tres justificaciones a nivel técnico para soportar la decisión final. Primero, su propuesta consiste en crear módulos con gestión de dependencias basados en Maven, con lo cual garantiza la estructuración del código en proyectos separados, lo que le permite separar la capa lógica, la capa de acceso a datos y la capa de servicios. Segundo, al utilizar Maven puede diseñar un arquetipo común, en el cual puede colocar lógica reutilizable, por ejemplo, acceso a métodos y clases comunes que se requieran desde diferentes microservicios, luego menciona que Maven le permite crear un arquetipo base para cada módulo. También anota una posible consecuencia en los microservicios que permite crear módulos en distintas tecnologías, lo que causa que el mantenimiento sea complejo. Una ventaja es que no es necesario detener la aplicación para realizar ajustes, cada componente funciona por separado. Además, documenta una táctica que consiste en autenticar a los clientes a través de tokens de seguridad JWT (JSON Web Tokens), generando un token único y usando un servicio de autenticación que garantice la validez de la conexión. Finalmente, anota una posible alternativa técnica que consiste en utilizar Laravel, un framework PHP.

6. Análisis

En el proyecto se emplearon diferentes estilos arquitectónicos motivados por diferentes decisiones de diseño. Un producto arquitectónico fue generado por cada componente durante las iteraciones realizadas. Diferentes grados de los detalles de las decisiones tomadas fueron registradas, para una mayor manipulación o proceso de mantenimiento. Tanto en el primer cambio arquitectónico como en el segundo tuvieron puntos de vista diferentes. La evaluación de la herramienta en ambos cambios demostró la utilidad de representar visualmente y gestionar el *rationale* arquitectónico de las decisiones de diseño. Hasta este punto, la interfaz de la herramienta fue suficiente para navegar fácilmente el fundamento y las decisiones tomadas. Por lo tanto, el arquitecto tiene una vista completa del proceso de diseño general, porque los elementos mostrados por la herramienta describen cronológicamente los eventos más importantes ocurridos durante el proyecto. Este enfoque facilita la comprensión del proceso de construcción de arquitectura para los interesados que están directamente relacionados con los problemas técnicos. Además, se genera documentación útil que incluye las decisiones tomadas, los patrones aplicados y los productos de arquitectura generados durante el proceso. La herramienta proporciona el modelo con la extensión *Rationale* Diagram que se asocia con la documentación nueva o existente, de la arquitectura de software como una vista más, en un contexto ágil. Aunque los ingenieros de software encontraron el lenguaje y la herramienta útil, se tienen algunos reparos al respecto. Primero, la calidad de la interface fue evaluada con solo el 72 %. Particularmente hay limitaciones para escalar el modelo de decisiones a todo el proyecto, la trazabilidad con los requisitos, en particular los no funcionales, así como su trazabilidad hacia el modelo arquitectónico expresado en otros modelos, como por ejemplo UML.

Es importante considerar que el caso se desarrolla en el contexto de una pequeña organización a través de varias sesiones de modelado, con un conjunto pequeño de decisiones de diseño (menos de tres). Sin embargo, la experiencia reportada por Zimmermann (2012), ha identificado 35 decisiones recurrentes en el desarrollo de aplicaciones empresariales en el contexto del diseño de arquitectura orientado a servicios. Además, Kellari, Crawley y Cameron (2018) analizan las decisiones arquitectónicas del software de aviones desde el DC3 hasta el 787, encontrando una disminución en la variación de las decisiones en un conjunto de datos, dee 157 arquitecturas en el histórico de aviones, llegando a la conclusión de que son 27 las decisiones arquitectónicamente relevantes. Estos dos escenarios; uno típico y otro extremo en términos de confiabilidad, permiten evidenciar que el lenguaje de modelado de decisiones y su *rationale* debe facilitar la formación del modelado de un número no pequeño de decisiones de diseño arquitectónico, facilitando una estructura de organización de dichas decisiones (relaciones temporales o causa/efecto), y un relacionamiento con elementos externos para facilitar su integración y trazabilidad. Un enfoque es visualizar decisiones de arquitectura desde diferentes perspectivas. Una perspectiva permitiría abordar un conjunto de preocupaciones relacionadas con la decisión. La idea de mostrar diferentes perspectivas de decisiones facilitaría la asociación directa con los puntos

de vista con los que describa la arquitectura, aunque esto traería más complejidad para relacionar. Además, tampoco hay una relación entre las decisiones de diseño y los artefactos de diseño arquitectónico donde estas decisiones se reflejan (vistas y modelos de la arquitectura).

Por otro lado, una efectiva toma de decisiones requiere de una comunicación efectiva y acciones coordinadas del equipo, lo cual agrega mayor complejidad a los proyectos open source, debido a que los desarrolladores de estos proyectos generalmente no trabajan en el mismo lugar (Harrison; Gubler; Skinner, 2016). Esto aumenta la posibilidad de decisiones arquitectónicas mal entendidas y la motivación detrás de estas. Además, es probable que existan objetivos diferentes, porque no necesariamente trabajan en la misma empresa y para los mismos intereses, por lo que la toma de decisiones en forma unificada es de vital importancia. En este escenario la herramienta DRML brindaría un mecanismo para eliminar la ambigüedad de las decisiones y la posibilidad de su unificación. Además del tema de la escalabilidad, en este escenario habría que explorar la posibilidad de brindarle capacidades de trabajo colaborativo asistido por computador. Y esto no solo resultaría beneficioso en proyectos open source, sino en general, porque el diseño arquitectónico requiere del trabajo en equipo que normalmente falla por varios motivos, como la falta de flexibilidad, el ego del personal y la lealtad hacia una tecnología preferida, evitando que se logre un consenso en las decisiones y su *rationale*. Algunos miembros del equipo constantemente intentan forzar su forma de hacer las cosas a los otros, en lugar de participar objetivamente en las discusiones (Dasanayake; Markkula; Aaramaa; Oivo 2016).

7. Conclusiones

En este artículo se evidencia la importancia de documentar el *rationale* arquitectónico de las decisiones de diseño como pieza clave en la documentación de la arquitectura de software, en un contexto ágil. Grabar y documentar el conocimiento de las decisiones de diseño de arquitectura es importante para los procesos de desarrollo y mantenimiento. Para lograr el objetivo en la documentación de la justificación de los cambios arquitectónicos, se ha propuesto un lenguaje y una herramienta basada en sus elementos especificados capaz de grabar, mantener y gestionar las decisiones tomadas durante el proceso de construcción y mantenimiento de la arquitectura.

Nuestro enfoque conecta los requisitos con las arquitecturas a través del *rationale* arquitectónico y decisiones de diseño, para establecer rastros entre ellos. El estudio indica que trabajar con MDE permite definir y realizar trasformaciones entre diferentes modelos, facilitando la interoperabilidad entre plataformas. Como un resultado de la evaluación realizada, se evidencia que este tipo de herramientas y, en particular, las herramientas basadas en meta-modelos, innovan la ingeniería del software con un enfoque centrado en desarrollo de aplicaciones a partir de modelos específicos de dominio de manera ágil y reduciendo costos, beneficiando a la industria del software y a la comunidad científica. También se identificaron limitaciones de la propuesta. Es necesario que el lenguaje permita especificar requerimientos funcionales y no funcionales y escalar el modelado para un número amplio de decisiones y sus relaciones. En cuanto a DRML, es importante que cuente con enlaces a fuentes externas de conocimiento, por ejemplo, UML, documentos de arquitectura y Wikis, entre otros. También sería útil generar un árbol de navegabilidad donde se pueda buscar el conocimiento documentado. Dado que el estudio es un caso exploratorio, este también tiene limitaciones en la muestra de los datos, ya que depende, en gran medida, de la experiencia y habilidades técnicas de los participantes en el conocimiento del tema para obtener mejores resultados. Como trabajo futuro se plantea extender el lenguaje para soportar el modelado de paquetes de decisiones (paquetes de preocupaciones y perspectivas), la trazabilidad con otros artefactos de la arquitectura, así como incorporar en la herramienta aspectos de versionado y modelado colaborativo. Además, aprovechando el paradigma MDE, relacionar por transformaciones los elementos de documentación del lenguaje DRML con las anotaciones de código desarrolladas por Dorado y Hurtado (2019), dado que el código sigue siendo la fuente más frecuente y confiable, en la que los desarrolladores encuentran información relevante a las justificaciones de las decisiones de diseño, particularmente, en el enfoque ágil con foco en la arquitectura.

Referencias

Aldrich, J.; Chambers, C.; y Notkin, D. (2002). ArchJava: connecting software architecture to implementation. *Proceedings of the 24th International Conference on Software Engineering*. ICSE 2002, (pp. 187–197). https://doi.org/10.1109/ICSE.2002.1007967

Bass, Len; Clements, Paul; Kazman, Rick (2003). *Software architectuSoftware architecture in practicere in practice*. Boston, EE.UU: Addison-Wesley.

Bucaioni, Alessio; Cicchetti, Antonio; Ciccozzi, Federico; Mubeen, Saad; Sjodin, Mikael (2017). A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL. *IEEE Access*, 5, 9005–9020. https://doi.org/10.1109/ACCESS.2016.2641218

Bunge, Mario (2002). *La investigación científica: su estrategia y su filosofía*. México: Siglo XXI Editores.

Che, Meiru (2014). *Managing Architectural Design Decision Documentation and Evolution Committee* (tesis doctoral). University of Texas at Austin, Austin, Texas.

Cleland-Huang, Jane; Mirakhorli, Mehdi; Czauderna, Adam; Wieloch, Mateusz (Mayo de 2013). Decision-Centric Traceability of architectural concerns. *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering, (TEFSE)*. San Francisco, California, EE. UU. https://doi.org/10.1109/TEFSE.2013.6620147

Dasanayake, Sandun; Markkula, Jouni; Aaramaa, Sanja; Oivo, Markku (2016). An Empirical Study on Collaborative Architecture Decision Making in Software Teams. In: Tekinerdogan B.; Zdun U.; Babar A. (eds). *European Conference on Software Architecture* (pp. 238-246). Springer, Cham: Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-319-48992-6_18

Dermeval, Diego; Castro, Jaelson; Silva, Carla; Pimentel, Joao; Bittencourt, Ibert; Brito, Patrick; Elias, Endhe; Tenorio, Thyago; Pedro, Alan (March de 2013). On the use of metamodeling for relating requirements and architectural design decisions. *SAC '13: Proceedings of the 28th Annual ACM Symposium on Applied Computing* (pp. 1278–1283). Coimbra, Portugal. https://doi.org/10.1145/2480362.2480601

Dorado, Santiago; Hurtado, Julio (2019). Documenting architectural *rationale* using source code annotations: An exploratory study. *EPiC Seriesin Computing*, 64, 204-214.

Gilson, Fabian; Englebert, Vincent (September de 2011). *Rationale*, decisions and alternatives traceability for architecture design. *ECSA 11: Proceedings of the 5th European Conference on Software Architecture: Companion* (pp. 1-9). New York, EEUU. https://doi.org/10.1145/2031759.2031764

Hadar, Irit; Sherman, Sofia; Hadar, Ethan; Harrison, Jhon (May 2013). Less is more: Architecture documentation for agile development. *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013 - Proceedings* (pp. 121–124). San Francisco, CA, USA. https://doi.org/10.1109/CHASE.2013.6614746

Harrison, Neil; Avgeriou, Paris (2010). How do architecture patterns and tactics interact? A model and annotation. *Journal of Systems and Software*, 83(10), 1735–1758. https://doi.org/10.1016/j.jss.2010.04.067

Harrison, Neil; Gubler, Erich; Skinner, Danielle (March de 2016). Architectural Decision-Making in Open-Source Systems-Preliminary Observations. *2016 1st International Workshop on Decision Making in Software ARCHitecture - Proceedings* (pp. 16–21). Venice, Italy.
https://doi.org/https://doi.org/10.1109/MARCH.2016.7

Hernández, Flor; Hurtado, Julio (2016). Difficulties and challenges in the incorporation of architectural practices. *Sistemas y Telemática*, 14(38), 74–86.
https://doi.org/10.18046/syt.v14i38.2290

Hesse, Tom-Michael; Kuehlwein, Arthur; Paech, Barbara; Roehm, Tobias; Bruegge, Bernd (2015). Documenting Implementation Decisions with Code Annotations. *SEKE*, 152-157.
https://doi.org/10.18293/SEKE2015-084

Jansen, Anton; Avgeriou, Paris; van Der Ven, Jan (2009). Enriching software architecture documentation. *Journal of Systems and Software*, 82(8), 1232–1248.
https://doi.org/10.1016/j.jss.2009.04.052

Jansen, A.; Bosch, J. (November 2005). Software architecture as a set of architectural design decisions. *5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005 – Proceedings (*pp. 109–120). Pittsburgh, Pennsylvania.
https://doi.org/10.1109/WICSA.2005.61

Kellari, Demetrios; Crawley, Edward; Cameron, Bruce (2018). Architectural decisions in commercial aircraft from the DC-3 to the 787. *Journal of Aircraft*, 55(2), 792–804.
https://doi.org/https://doi.org/10.2514/1.C034130

Kruchten, Philippe; Capilla, Rafael; Dueñas, Juan (2009). The decision view's role in software architecture practice. *IEEE Software*, 26(2), 36–42.
https://doi.org/10.1109/MS.2009.52

Kruchten, Philippe; Obbink, H.; Stafford, J. (2006, March). The past, present, and future for software architecture. IEEE Software, 23(2), 22-30.
https://doi.org/10.1109/MS.2006.59

Lopes, Socrates; Aquino, Plinio (2017). Architectural Design Group Decision-Making in Agile Projects. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*(pp. 201-2015). Gothenburg, Sweden.
https://ieeexplore.ieee.org/abstract/document/7958489/

Malavolta, Ivano; Muccini, Henry; Rekha, Smrithi (2014). Enhancing Architecture Design Decisions Evolution with Group Decision Making Principles. In: Majzik I., Vieira M. (eds). *Software Engineering for Resilient Systems. SERENE 2014. Lecture Notes in Computer Science* (pp. 9–23). Budapest, Hungary: Springer.
https://doi.org/10.1007/978-3-319-12241-0_2

Manteuffel, Christian; Tofan, Dan; Koziolek, Heiko; Goldschmidt, Thomas; Avgeriou, Paris (2014). Industrial implementation of a documentation framework for architectural decisions. *2014 IEEE/IFIP Conference on Software Architecture – Proceedings* (pp. 225–234). Sydney, NSW.
https://doi.org/https://doi.org/10.1109/WICSA.2014.32

Montenegro, Carlos; Gaona, Paulo; Cueva, Juan; San Juan, Oscar (2011). Aplicación de ingeniería dirigida por modelos (MDA), para la construcción de una herramienta de modelado de dominio específico (DSM) y la creación de módulos en sistemas de gestión de aprendizaje (LMS) independientes de la plataforma. *Dyna*, 78 (169), 43–52.
https://www.redalyc.org/pdf/496/49622390005.pdf

Plataniotis, Georgios; Ma, Qin; Proper, Erik; de Kinderen, Sybren (2015). Traceability and modeling of requirements in enterprise architecture from a design *rationale* perspective. *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)* (pp. 518–519). Athens, Greece. https://doi.org/10.1109/RCIS.2015.7128916

Reynoso, Carlos (2004). *Introducción a la Arquitectura de Software.* http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:introarq.pdf

Roeller, Ronny; Lago, Patricia; van Vliet, Hans (2006). Recovering architectural assumptions. *Journal of Systems and Software*, 79(4), 552–573. https://doi.org/10.1016/j.jss.2005.10.017

Roldán, Maria; Gonnet, Silvio; Leone, Horacio (2016). Operation-based approach for documenting software architecture knowledge. *Expert Systems*, 33(4), 313–348. https://doi.org/10.1111/exsy.12152

Runeson, Per; Höst, Martin (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2), 131–164. https://doi.org/10.1007/s10664-008-9102-8

Sauro, Jeff; Lewis, James (2016). Standardized usability questionnaires. In *Quantifying the User Experience* (pp. 185–248). EE.UU: Elsevier. https://doi.org/10.1016/b978-0-12-802308-2.00008-4

Singer, J. (1998). Practices of software maintenance. *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)* (pp. 139–145). Bethesda, EE.UU. https://doi.org/10.1109/ICSM.1998.738502

Tang, Antony; Babar, MMuhammad; Gorton, Ian; Han, Jun (2006). A survey of architecture design *rationale*. *Journal of Systems and Software*, 79(12), 1792–1804. https://doi.org/10.1016/j.jss.2006.04.029

van Der Ven, Jan; Bosch, Jan (2016). Busting Software Architecture Beliefs: A Survey on Success Factors in Architecture Decision Making. *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 42–49). Limassol, Cyprus. https://doi.org/10.1109/SEAA.2016.35

van Der Ven, Jan; Jansen, Anton; Nijhuis, Jos; Bosch, Jan (2006). Design decisions: The bridge between *rationale* and architecture. In Dutoit A.H., McCall R., Mistrík I., Paech B. (eds). *Rationale Management in Software Engineering* (pp. 329–348). Berlin, Heidelberg, Springer. https://doi.org/https://doi.org/10.1007/978-3-540-30998-7_16

van Vliet, Hans; Tang, Antony (2016). Decision making in software architecture. *Journal of Systems and Software*, 117, 638–644. https://doi.org/10.1016/j.jss.2016.01.017

Zimmermann, Olaf (2012). Architectural decision identification in architectural patterns. In: *WICSA/ECSA '12: Proceedings of the WICSA/ECSA 2012 Companion Volume* (pp. 96–103). New York, EE.UU: Association for Computing Machinery. https://doi.org/10.1145/2361999.2362021