RADS: Una herramienta para reutilizar estrategias en diseños de arquitecturas de software.

María Celeste Carignano, Silvio Gonnet, Horacio Leone

Universidad Tecnológica Nacional, Facultad Regional Santa Fe - INGAR, Instituto de Desarrollo y Diseño
mcarigna@frsf.utn.edu.ar, sgonnet@santafe-conicet.gov.ar, hleone@santafe-conicet.gov.ar

Abstract. El diseño de arquitecturas de software es un proceso altamente creativo que aún no ha sido estandarizado, por lo que las actividades llevadas a cabo para construir la arquitectura de un sistema son aquellas que los arquitectos involucrados consideran convenientes y pertinentes según el método de diseño que utilicen, su experiencia, conocimientos y habilidades personales. La reutilización es una práctica habitual dentro de dicha actividad. Sin embargo, no existen herramientas que asistan a los arquitectos para llevarla a cabo. En este trabajo se describe una herramienta de software construida con el objetivo de colaborar con los arquitectos de software en la recuperación de experiencias pasadas, propias y ajenas, para hacerlas accesibles de manera de que puedan ser reutilizadas durante el diseño de un nuevo sistema.

Keywords. Captura, recuperación, reutilización, tácticas, estilos, decisiones de diseño, arquitecturas de software, razonamiento basado en casos, herramienta.

1 Introducción

El diseño de arquitecturas de software es un proceso altamente creativo que aún no ha sido estandarizado, por lo que las actividades llevadas a cabo para construir la arquitectura de un sistema son aquellas que los arquitectos involucrados consideran convenientes y pertinentes según el método de diseño que utilicen, su experiencia, conocimientos y habilidades personales. Según Kruchten toda arquitectura de software se crea a partir de tres fuentes: método, intuición, y reutilización [1]. De alguna manera, la esencia del trabajo de un arquitecto de software es encontrar y aplicar un balance correcto entre estas tres fuentes de una arquitectura [2]. La proporción en que estas fuentes influyen en un diseño arquitectónico varía de acuerdo a la experiencia del arquitecto y al grado de novedad del sistema a ser diseñado [1].

La reutilización o reuso es el proceso en el cual productos de trabajo de software existentes (como ser código, documentación, diseño, datos de prueba, herramientas y especificaciones) son utilizados en nuevos desarrollos, preferentemente con modificaciones mínimas [3]. En este contexto, el arquitecto debe encontrar alguna analogía entre parte o la totalidad del problema planteado para el sistema que intenta construir y los problemas que originaron sistemas ya existentes que él conoce. La

analogía es un proceso cognitivo sofisticado en el cual dos situaciones, una origen y una destino, son analizadas para encontrar patrones estructurales comunes ([4], [5]) con el objetivo de aplicar el conocimiento disponible acerca de la situación origen en la situación destino una vez que se hayan efectuado las adaptaciones necesarias.

En la práctica, en una encuesta realizada a arquitectos de software de Argentina [6], se pudo observar que el 98.5% de los encuestados hacía uso de la reutilización durante el diseño arquitectónico. Por lo que puede ser catalogada como una práctica habitual dentro de la actividad de diseño. Sin embargo, no existen herramientas que asistan a los arquitectos para llevarla a cabo.

Ante esta situación, se ha identificado la necesidad de contar con una herramienta que pueda colaborar con los arquitectos de software en la recuperación de experiencias pasadas, propias y ajenas, con el objetivo de hacerlas accesibles de manera de que puedan ser reutilizadas para dar solución al diseño de un nuevo sistema.

El propósito de dicha herramienta no estaría centrado en intentar desplazar o reemplazar al arquitecto ni automatizar por completo su trabajo, sino que estaría dirigido a facilitar su labor y la de todos aquellos que comparten experiencias pasadas que pueden ser de utilidad para la construcción de nuevos sistemas.

En este trabajo se presentan los detalles de la herramienta construida, llamada RADS, para cumplir con los objetivos planteados. A continuación, en la Sección 2 se realiza una breve descripción del marco teórico sobre el cual se basa la herramienta. En la Sección 3, se enumeran algunas de las principales características y se presenta un bosquejo de la arquitectura de RADS. En la Sección 4, se describen las principales funcionalidades brindadas por la herramienta. Y finalmente, en la Sección 5, se presentan las conclusiones y los trabajos futuros.

2 Marco teórico de RADS

RADS es una herramienta que ha sido creada para lograr el objetivo planteado, permitiendo:

- capturar las experiencias de los arquitectos de una forma estructurada, de manera tal que brinda la posibilidad de efectuar comparaciones entre ellas,
- almacenar dichas experiencias en un repositorio común y compartido,
- recuperar las experiencias almacenadas en el repositorio a partir de la identificación de una nueva experiencia,
- articular los medios necesarios para que las soluciones empleadas en las experiencias recuperadas sean aplicadas en la nueva experiencia de diseño arquitectónico a llevar a cabo.

El marco teórico en el cual se centra la herramienta RADS está descrito en la Tesis Doctoral "Representación y razonamiento sobre las decisiones de diseño de arquitectura de software" [7]. En dicha Tesis se aborda el problema identificado, proponiendo como solución una metodología que aplica la técnica de Razonamiento Basado en Casos en el contexto del diseño arquitectónico.

Razonamiento Basado en Casos es un paradigma de resolución de problemas [8] que involucra el uso de experiencias pasadas para comprender y resolver nuevas situaciones [9]. Un caso denota una situación experimentada previamente, la cual ha sido capturada y aprendida de manera que pueda ser reutilizada en la resolución de problemas futuros [8]. Suele estar compuesto por: (a) el problema: que describe el estado del mundo cuando ocurre el caso, y (b) la solución: que establece la solución encontrada.

En el contexto de RADS, un caso describe la arquitectura diseñada para un sistema de software en particular y es llamado *caso arquitectónico*. Los casos arquitectónicos están estructurados en base al estándar internacional ISO/IEC/IEEE 42010:2011 [10] y se componen de dos partes:

- problema: identifica las necesidades que el sistema debe satisfacer mediante el empleo de restricciones de diseño y una representación particular de los requerimientos de calidad materializada por medio de e*scenarios de atributos de calidad* (adoptados del trabajo de Bass y otros [11]). Estos últimos son utilizados para documentar de manera organizada los requerimientos de calidad de los "stakeholders". Dos tipos de escenarios son empleados: los escenarios generales y los escenarios específicos del sistema [12][11]. Se excluyen en este punto los requerimientos funcionales por considerar que el objetivo de ***RADS*** es brindar soporte al arquitecto en la definición temprana de la arquitectura y siendo que en dicho estadio los requerimientos funcionales no ejercen tanta influencia [13]. De esta manera, no se da lugar a la documentación de información que posteriormente no será utilizada.
- solución: describe la resolución del problema en términos que faciliten su reutilización en nuevos casos. En el contexto de [7], la reutilización se lleva a cabo en el nivel del conocimiento [14]. Los conceptos involucrados son los de decisiones de diseño arquitectónico y estrategias de diseño (estilos [11] [15] y tácticas arquitectónicas [16]).

La técnica de razonamiento basado en casos involucra la ejecución de cuatro actividades conocidas como las 4Res [8] (*R*ecuperar, *R*eusar, *R*evisar y *R*etener), las cuales son presentadas en la Fig. 1.

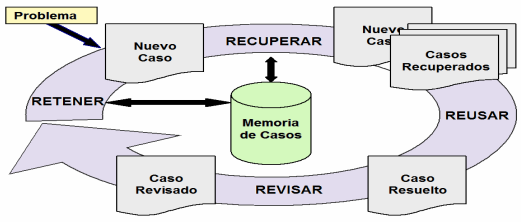

Fig. 1. Actividades del proceso de Razonamiento Basado en Casos. Figura adaptada de [8].

La primera actividad es la de **RECUPERACIÓN** de los casos más similares. Comienza con la descripción de un problema (conocido como nuevo caso) y finaliza con la obtención de uno o más casos recuperados. Cada caso recuperado propone soluciones previamente aplicadas para resolver un problema similar. Involucra la evaluación de todos los casos en memoria y la determinación del grado de similitud entre cada uno de ellos y el nuevo caso.

Durante la actividad de recuperación definida en ***RADS*** se llevan a cabo varios procesamientos que involucran al nuevo caso y los casos de la memoria. Como primer paso se verifica que las restricciones de diseño establecidas en el nuevo caso sean satisfechas en el contexto de las decisiones de diseño documentadas en los casos almacenados en la memoria. Aquellos casos de la memoria que cumplen con las restricciones obligatorias del nuevo caso son seleccionados para continuar con el segundo paso del procesamiento, en el cual se establece la similitud existente entre el nuevo caso y los casos de la memoria en función de los escenarios específicos del sistema y las prioridades de los mismos. Solo aquellos casos de la memoria que cumplen con las restricciones de diseño obligatorias y tienen un valor de similitud importante (establecida mediante un parámetro) son devueltos como resultado de esta actividad.

La segunda actividad tiene como objetivo **REUSAR** la información y el conocimiento del caso recuperado para resolver el nuevo problema. Es la responsable de proporcionar una solución (conformando un caso resuelto) para un nuevo problema a partir de las soluciones del caso recuperado: soluciones antiguas son utilizadas como inspiración para resolver nuevos problemas [9]. En ***RADS*** esta actividad no es automática, sino que requiere la intervención de los arquitectos de software ya que la reutilización se realiza en el nivel del conocimiento aplicado en el caso de la memoria como base para construir una propuesta inicial de solución del nuevo caso. Sobre dicha propuesta inicial deberán trabajar los arquitectos involucrados en el diseño arquitectónico para definir la arquitectura completa del sistema en cuestión.

La tercera actividad es la de **REVISIÓN** de la solución propuesta. Una vez que el nuevo caso ha sido resuelto se deben llevar a cabo evaluaciones y verificaciones sobre la solución alcanzada. Cuando la solución generada por la actividad de reuso no es correcta surge una oportunidad para aprender de las fallas. De esta forma, durante la actividad de revisión, se llevan a cabo dos tareas [8]: la evaluación del caso resuelto generado por la actividad de reuso, y la reparación del caso resuelto, si es necesario.

Durante la última actividad, la de **RETENCIÓN**, el caso revisado es incorporado a la memoria de casos como un caso aprendido para que esté disponible si un nuevo problema arriba para comenzar con el ciclo nuevamente.

3 Requerimientos y arquitectura de RADS

Los principales requerimientos funcionales de RADS están vinculados a:

- la captura de información relacionada a los intereses expresados por los "stakeholders" de un sistema mediante escenarios específicos del sistema y restricciones de diseño,

- la documentación de las decisiones tomadas por los arquitectos que diseñan la arquitectura del sistema en cuestión, y
- la recuperación de las estrategias de diseño aplicadas previamente para emplearlas en nuevos sistemas.

Dichos requerimientos están condicionados por los siguientes aspectos:

- integración con herramientas de modelado: debido a que no existe una herramienta estandarizada para realizar el diseño de arquitecturas de software, cada arquitecto puede emplear una diferente, por lo que ***RADS*** debe integrarse con cualquier herramienta de modelado UML para facilitar y propiciar su utilización. Surge así la necesidad de contar con una solución arquitectónica que involucre la menor cantidad de cambios posibles cada vez que la aplicación deba integrarse con una nueva herramienta de modelado,
- todas las instancias que se ejecuten de ***RADS*** deben compartir un repositorio común y único en el cual los arquitectos pueden volcar sus experiencias para formar una memoria corporativa de casos arquitectónicos,
- ***RADS*** debe brindar la posibilidad de que un caso arquitectónico sea documentado desde más de una instancia de la aplicación para no dificultar u obstruir la realización de diseños colaborativos entre varios arquitectos.

Además, teniendo en cuenta los resultados obtenidos de los trabajos de Tang y otros [17] y Carignano y otros [6], en donde se puede apreciar que durante un diseño arquitectónico uno de los recursos más limitado para los arquitectos es el tiempo; para que RADS sea de utilidad debe cumplir con dos requerimientos de calidad:

- debe ser fácil de usar para no entorpecer el trabajo de los arquitectos, y
- no debe requerir mayor esfuerzo del que lleva documentar correcta y completamente una arquitectura, considerando que dentro de la definición de arquitectura también se incluye el conocimiento arquitectónico involucrado.

Una versión simplificada de la arquitectura diseñada para satisfacer los requerimientos especificados se presenta en la Fig. 2.

RADS se estructura en cuatro capas: presentación, lógica, persistencia, y utilidades. La selección de este estilo arquitectónico estuvo motivada por la necesidad de facilitar la integración con herramientas de modelado UML ya existentes: la capa presentación puede ser reemplazada o eliminada según se requiera sin afectar al resto de la aplicación. Por definición, el estilo de capas refleja una división del software en unidades o capas, y cada una de dichas unidades agrupa de forma lógica varios módulos que ofrecen un conjunto cohesivo de servicios a otras capas [18]. De esta forma: la *capa lógica* presta servicios a la capa de presentación por medio de la utilización de fachadas (*Facades* en Fig. 2), la *capa de persistencia* presta servicios a la capa lógica por medio de objetos de acceso a datos (*Data Access Objects* en Fig. 2), y la *capa de utilidades* presta servicios a todas las capas al poner a disposición los objetos de intercambio de información (*Data Transfer Objects* en Fig. 2) y los objetos persistentes (*Domain Objects* en Fig. 2).

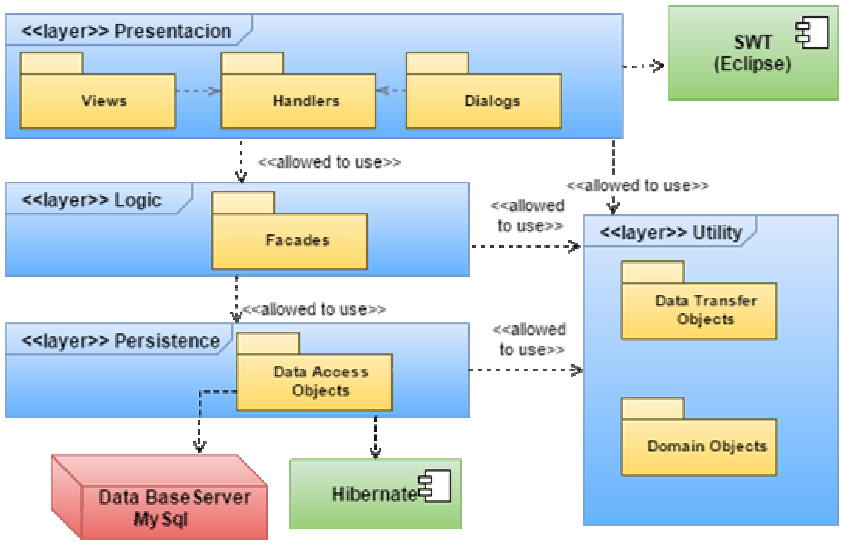

Fig. 2. Versión simplificada de la arquitectura de ***RADS***.

RADS ha sido implementada en *JAVA* [19] como un complemento ("plug-in") de *Eclipse* [20] utilizando *SWT* (siglas en inglés de Standard Widget Toolkit [21]) para construir la interfaz gráfica de la aplicación. De esta manera, ***RADS*** se integra a entornos de herramientas de modelado UML ya existentes, como ser: IBM Rational Software Architect, Papyrus, UML2, entre otros.

Los casos son almacenados en una base de datos relacional: *MySQL* [22], por lo que ***RADS*** hace uso de *Hibernate* [23] para llevar a cabo el mapeo objeto-relacional de los datos manipulados. De esta forma, se satisface el requisito de contar con un repositorio centralizado de casos arquitectónicos, ya que varias instancias de ***RADS*** pueden acceder a la base de datos seleccionada.

4 Principales características funcionales de la herramienta

La versión simplificada del modelo de casos de usos presentado en la Fig. 3 permite conocer las funcionalidades ofrecidas por la herramienta **RADS**.

Se identifican tres actores que pueden relacionarse con el sistema. El *administrador RADS* es el responsable de crear inicialmente, y mantener posteriormente, información que es común a todos los sistemas: atributos de calidad, escenarios generales y estrategias de diseño (tácticas y estilos arquitectónicos). El analista es el responsable de introducir al sistema los intereses refinados explicitados por los "stakeholders": escenarios específicos del sistema y restricciones de diseño. Y el arquitecto: responsable de capturar el razonamiento asociado a la descripción arquitectónica vinculada al sistema de interés por medio de decisiones de diseño y capaz de consultar por experiencias pasadas para su reutilización.

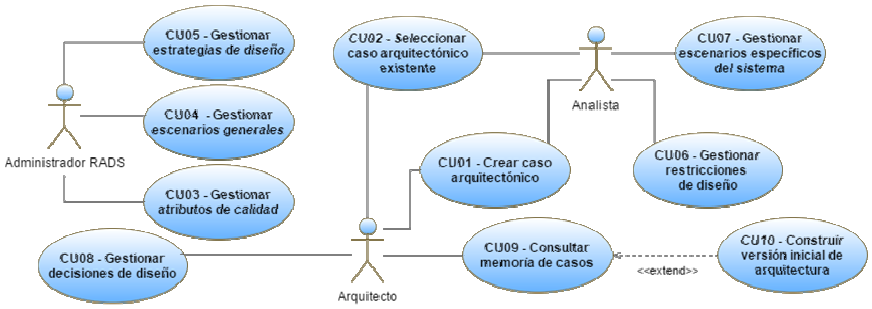

Fig. 3. Versión simplificada del modelo de casos de usos de RADS.

El entorno de trabajo de *Eclipse* consiste de varios paneles conocidos como *vistas*. **RADS** hace uso de ellas (ver Fig. 4) para presentar la información de los principales conceptos: atributos de calidad, escenarios generales, estrategias de diseño, escenarios específicos del sistema, restricciones de diseño, y decisiones arquitectónicas.

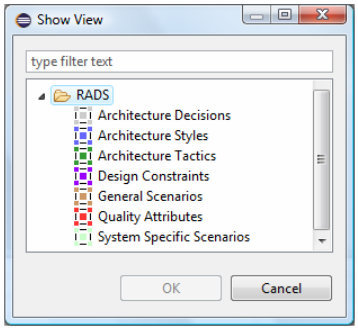

Fig. 4. Vistas de Eclipse pertenecientes al complemento ***RADS***

En ***RADS*** existen algunos conceptos cuyas instancias son empleadas por todos los casos arquitectónicos almacenados en un contexto de aplicación dado: los atributos de calidad, los escenarios generales y las estrategias de diseño.

RADS brinda la posibilidad de que un *Administrador RADS* pueda gestionar cada uno de estos conceptos (Casos de Uso *CU03*, *CU04*, y *CU05* en Fig. 3). Se incluyen dentro de la definición de "gestión" a todas aquellas acciones que permiten mantener actualizadas las instancias de los conceptos involucrados, es decir: el acceso, la creación, la modificación, y la eliminación, en caso de ser necesario.

El acceso a los datos se realiza mediante las vistas definidas para cada caso. En la Fig. 5 se presentan las vistas correspondientes a las entidades compartidas mencionadas previamente.

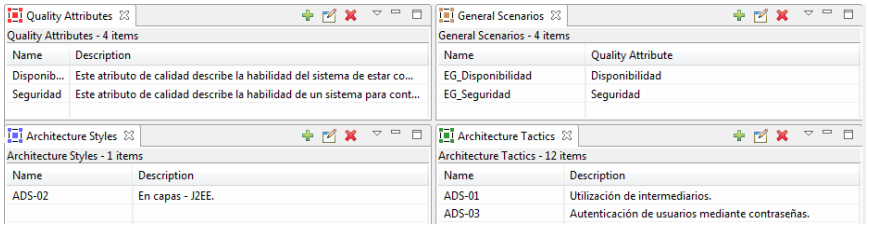

Fig. 5. Vistas de ***RADS*** correspondientes a atributos de calidad, escenarios generales, tácticas arquitectónicas y estilos arquitectónicos.

La creación, modificación y eliminación de los datos puede iniciarse desde la vista correspondiente. Cada entidad tiene asociada diálogos específicos de creación y modificación, cuyas características dependen de los datos de dicha entidad. Por ejemplo, en la Fig. 6 se muestran los diálogos utilizados para capturar la información relacionada con escenarios generales.

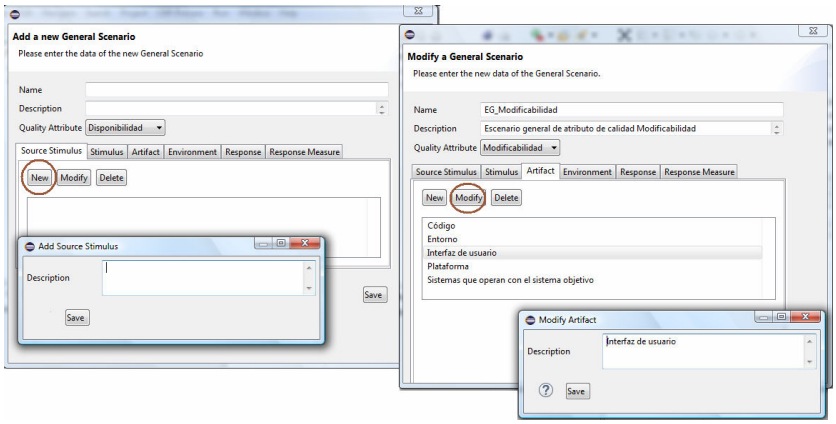

Fig. 6. Creación y modificación de escenarios generales

Antes de comenzar con la captura de la información de un sistema en particular, se debe ingresar dicho sistema al contexto de la aplicación ***RADS***. Esto es logrado al asociar cualquier proyecto de *Eclipse* a un caso arquitectónico indicando su nombre desde la ficha de propiedades de dicho proyecto, tal como lo indica la Fig. 7.

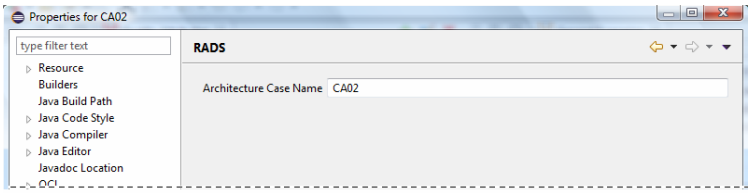

Fig. 7. Crear y asociar proyecto a caso arquitectónico en el contexto de ***RADS***.

Si el caso arquitectónico no existe, se crea uno nuevo (*CU01* en Fig. 3). En cambio, si ya se ha registrado un caso arquitectónico con el nombre proporcionado se vincula con el proyecto seleccionado (*CU02* en Fig. 3). De esta manera, se permite trabajar con un mismo caso arquitectónico desde más de un entorno de trabajo de *Eclipse*.

A partir de que un caso arquitectónico ha sido creado, puede capturarse información relacionada a los intereses expresados por los "stakeholders" de un sistema, es decir, las restricciones de diseño (*CU06* en Fig. 3) y los escenarios específicos del sistema (*CU07* en Fig.3). En la Fig. 8 se presentan las vistas de estas entidades.

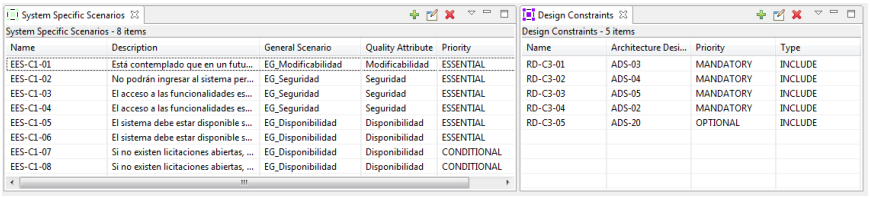

Fig. 8. Vista de ***RADS***: escenarios específicos del sistema y restricciones de diseño.

Una vez definidos los intereses de los "stakeholders", se pueden capturar las decisiones de diseño tomadas por los arquitectos para satisfacerlos (*CU08* en Fig. 3). En la Fig. 9 se presenta la vista correspondiente a esta entidad.

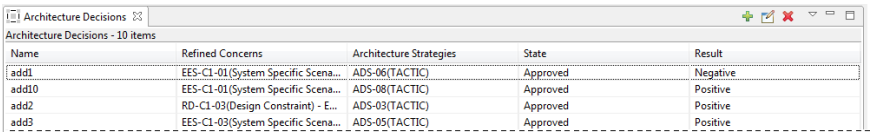

Fig. 9. Vista de ***RADS*** correspondiente a decisiones de diseño arquitectónico.

Finalmente, ***RADS*** implementa la técnica de razonamiento basado en casos descrita, por lo que los arquitectos pueden consultar las estrategias de diseño propuestas para diseñar casos arquitectónicos nuevos en base a los casos arquitectónicos documentados en la base de casos (*CU09* en Fig. 3).

Por cada caso recuperado, la principal información presentada consta de:

- las estrategias de diseño recomendadas (ver Fig. 10). Para cada escenario del sistema del nuevo caso, para el cual existe alguna estrategia de diseño para recomendar, se describen las distintas alternativas de estrategias de diseño recomendadas.

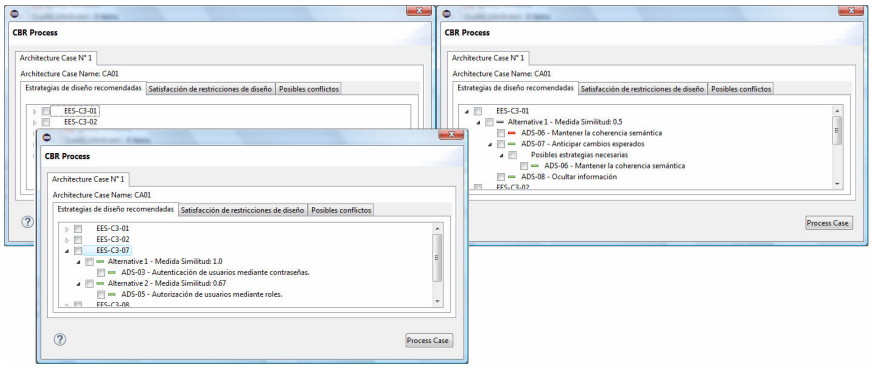

Fig. 10. Estrategias de diseño recuperadas.

- una descripción de cómo se satisfacen las restricciones de diseño, indicándose cuáles se cumplen y cuáles no (ver Fig. 11).

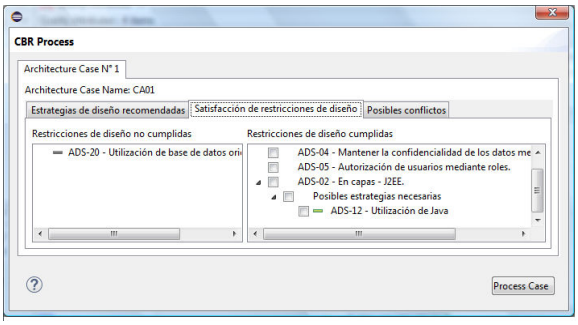

Fig. 11. Satisfacción de las restricciones de diseño.

A partir de la información presentada, el arquitecto puede construir una versión inicial de la descripción arquitectónica compuesta por las estrategias de diseño seleccionadas para satisfacer determinados escenarios específicos del sistema o restricciones de diseño. Para ello debe tildar aquellas estrategias de diseño que desea aplicar e indicar que desea el procesamiento de su petición, de manera que se generen automáticamente las decisiones de diseño correspondientes (ver Fig. 12).

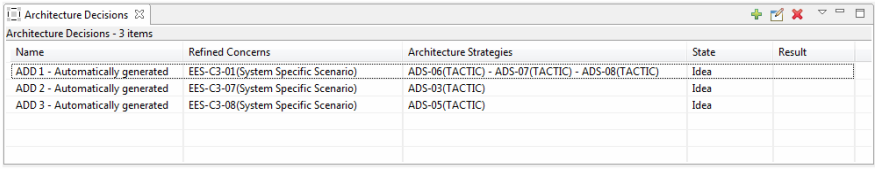

Fig. 12. Resultado de la reutilización de decisiones de diseño

5 Conclusiones y trabajos futuros

En este trabajo se presentaron las características generales de una herramienta, llamada ***RADS***, construida para dar soporte a los arquitectos de software durante el diseño arquitectónico. Se describieron los requerimientos identificados, tanto de calidad como funcionales, y los principales aspectos de la arquitectura diseñada para satisfacerlos.

RADS está basada en los resultados documentados en la Tesis Doctoral "Representación y razonamiento sobre las decisiones de diseño de arquitectura de software" [7]. En [7] se establecen como pilares a la técnica de Razonamiento Basado en Casos para determinar las actividades llevadas a cabo durante el proceso de reutilización; y el estándar internacional ISO/IEC/IEEE 42010:2011 para definir los principales conceptos involucrados.

El objetivo de ***RADS*** es brindar a los arquitectos información sobre experiencias de diseño arquitectónico pasadas (almacenadas en una memoria de casos) que presenten algún tipo significativo de similitud con el nuevo sistema para el cual debe definirse la arquitectura. Para ello previamente debe permitir la captura de información relacionada con la descripción del problema de los casos arquitectónicos que intervienen, es decir, con las restricciones de diseño y los escenarios específicos del sistema; y con la descripción de la solución de los casos arquitectónicos pertenecientes a la memoria de casos, es decir, las decisiones arquitectónicas definidas.

La herramienta presentada no intenta automatizar la totalidad del trabajo de los arquitectos de software, sino que persigue como finalidad: brindar soporte a su trabajo en las etapas tempranas de la definición de una arquitectura, que es cuando se toman las principales decisiones de diseño que definirán los lineamientos generales del nuevo sistema.

Como trabajo futuro, se plantea el enriquecimiento de los algoritmos de recuperación de manera que se tengan en cuenta otros elementos que contribuyen a la definición de un sistema de software y la calidad del proyecto de desarrollo en si como son las métricas de diseño. Así como también, la integración de ***RADS*** con nuevas herramientas de diseño UML.

Agradecimientos

Este trabajo ha sido financiado en forma conjunta por la Universidad Tecnológica Nacional y CONICET. Se agradece el apoyo brindado por estas instituciones.

Referencias

[1] P. Kruchten. Mommy, where do software architecture come from? 1st International Workshop on Architectures for Software Systems (IWASS1), pages 198–205, 1995.

[2] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, 2010

[3] T. B. Bollinger and S. L. Pfleeger. The economics of reuse: Issues and alternatives. In GA Atlanta, editor, Proceedings of the Eighth Annual National Conference on Ada Technology, pages 436–447, 1990.
[4] D. Gentner. Similarity and analogical reasoning. chapter The Mechanisms of Analogical Learning, pages 197–241. Cambridge University Press, 1989.
[5] H. Gust, U. Krumnack, K. U. Kuhnberger, and A. Schwering. Analogical¨ reasoning: A core of cognition. Zeitschrift fur Künstliche Intelligenz (KI),¨ Themenheft KI und Kognition, (1):8–12, 2008.
[6] M. C. Carignano, S. Gonnet, and H. Leone. Reasoning and Reuse in Software Architecture Design: Practices in the Argentine Industry. SADIO Electronic Journal of Informatics and Operations Research, 12(1), September 2013.
[7] M. C. Carignano. Representación y razonamiento sobre las decisiones de diseño de arquitectura de software. Tesis doctoral. ISBN 978-987-33-8486-8, 2015
[8] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications. IOS Press, 7(1):39–59, 1994.
[9] J. Kolodner. An introduction to case-based reasoning. Artificial Intelligence Review, 6:3–34, 1992.
[10] International Organization for Standardization and International Electrotechnical Commission. Systems and Software Engineering - Architecture Description (ISO/IEC/IEEE 42010). Edition 01-12-2011. 2011.
[11] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice, Second Edition. Addison-Wesley, 2003.
[12] L. Bass, M. Klein, and G. Moreno. Applicability of general scenarios to the architecture tradeoff analysis method. TR CMU/SEI-2001-TR-014, SEI, 2001.
[13] Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice, Third Edition. Addison-Wesley, 2013.
[14] R. Prietro-Díaz and P. Freeman. Classifying software for reusability. IEEE Software, 4(1):6, 1987.
[15] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, Robert Nord, and Judith Stafford. Documenting Software Architectures: Views and Beyond (2nd Edition). Addison-Wesley, 2010
[16] F. Bachmann, L. Bass, and M. Klein. Illuminating the fundamental contributors to software architecture quality. TR CMU/SEI-2002-TR-025, SEI, 2002.
[17] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han. A survey of architecture design rationale. The Journal of Systems and Software, Elseiver, 79:1792–1804, 2006.
[18] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, P. Merson, Robert Nord, and Judith Stafford. Documenting Software Architectures: Views and Beyond (2nd Edition). Addison-Wesley, 2010.
[19] Oracle Corporation. Java. [Web - www.oracle.com/technetwork/java/index.html; accedido el 05-05-2016].
[20] Eclipse Foundation. Eclipse. [Web - www.eclipse.org; accedido el 05-05-2016]
[21] Eclipse Foundation. Standard widget toolkit. [Web - www.eclipse.org/swt; accedido el 05-05-2016]
[22] Oracle Corporation. Mysql. [Web - www.mysql.com; accedido el 05-05-2016].
[23] Red Hat. Hibernate. [Web - hibernate.org; accedido el 05-05-2016]