Representación y razonamiento sobre las decisiones de diseño de arquitectura de software.

Autor: María Celeste Carignano
Director: Dr. Horacio Leone
Codirector: Dr. Silvio Gonnet
Fecha de exposición: 1 de Julio de 2015

Tesis presentada para la obtención del grado de Doctor en Ingeniería, mención Sistemas de Información, Facultad Regional Santa Fe, Universidad Tecnológica Nacional.

1. Introducción

La arquitectura de un sistema puede ser vista como las estructuras del sistema que comprenden a los elementos de software, las propiedades visibles externamente de dichos elementos, y las relaciones entre ellos [1]. En las últimas décadas las arquitecturas de software han ido tomando un protagonismo creciente dentro del ciclo de vida de un proyecto de software. Este protagonismo se fue incrementando de la mano de la complejidad de los sistemas construidos.

En la actualidad, construir sistemas sin diseñar previamente su arquitectura es equivalente a construir edificios sin definir previamente sus planos.

La importancia de las arquitecturas de software no solo radica en que permiten capturar la o las estructuras de un sistema, sino que también ([1], [2]):

- son un vehículo de comunicación entre los "stakeholders",
- sirven como un mecanismo de educación y entrenamiento,
- manifiestan las decisiones tempranas sobre los sistemas,
- se constituyen como abstracciones transferibles de sistemas diseñados,
- sirven como base para el análisis de los sistemas y sus construcciones,
- facilitan el razonamiento frente a cambios en los sistemas, y
- permiten predecir características de calidad del sistema a partir de su estudio.

Si bien el diseño de arquitecturas de software tiene un costo asociado al tiempo, el esfuerzo y los recursos utilizados para tal fin, el no contar con una arquitectura completamente diseñada tiene un costo mayor [2] e impacta en la calidad del producto resultante, el tiempo insumido para el desarrollo del sistema e incluso en la culminación con éxito del proyecto de desarrollo.

El diseño de arquitecturas de software es un proceso altamente creativo que aún no ha sido formalizado, 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 [3] toda arquitectura de software se crea a partir de tres fuentes:

- método: éste puede ser visto como una manera sistemática, concisa y documentada mediante la cual la arquitectura es derivada desde los requerimientos del sistema y las restricciones tecnológicas;
- intuición: considerada como la habilidad de concebir sin razonamiento consciente;
- reutilización: la mayoría de los elementos de una arquitectura de software son obtenidos de otras arquitecturas, principalmente cuando los arquitectos se encuentran familiarizados con un sistema previo del mismo tipo, otro sistema con características similares, o alguna arquitectura encontrada en la literatura técnica.

Kruchten [3] opina que la proporción en que estas tres fuentes influyen en un diseño arquitectónico varía de acuerdo a la experiencia del arquitecto y al grado de novedad del sistema a ser diseñado. Por ejemplo, en sistemas clásicos la reutilización de soluciones prima por sobre la intuición, ya que se tienen como referencias las arquitecturas definidas para sistemas con los cuales se ha trabajado previamente (Fig. 1 (a)). Mientras que en sistemas sin precedentes, la intuición juega un papel importante (Fig. 1 (b)). Sin embargo, en ambos casos el proceso de construcción de

la arquitectura debe estar soportado, en mayor o menor medida, por algún método de diseño validado. Las arquitecturas que se construyen sin emplear un método están destinadas al fracaso.

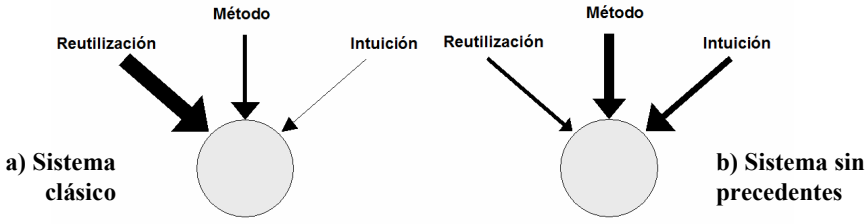

Figura 1: Fuentes de un diseño arquitectónico, extraído de [3]

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 [4].

En lo que respecta a métodos de diseño, el arquitecto cuenta con una amplia variedad de ellos. En las últimas décadas se han definido y documentado diferentes métodos, procesos y técnicas de diseño de arquitecturas de software. Debido a que muchos de ellos fueron desarrollados de manera independiente, difieren en diversos aspectos, incluso en los términos que utilizan.

Hofmeister y otros [5] efectuaron una comparación de cinco de los métodos de diseño de arquitecturas utilizados en la industria de software. A partir de esta comparación, se identificaron varios puntos en común entre ellos y algunas diferencias. Como principal aporte resultante de dicho trabajo se definió un proceso de diseño arquitectónico genérico que describe las actividades llevadas a cabo por los arquitectos: análisis, síntesis, y evaluación arquitectónica (Fig. 2). Estas actividades no se realizan de manera secuencial sino que forman parte de un proceso iterativo en el cual los artefactos van evolucionando a medida que éstas son ejecutadas.

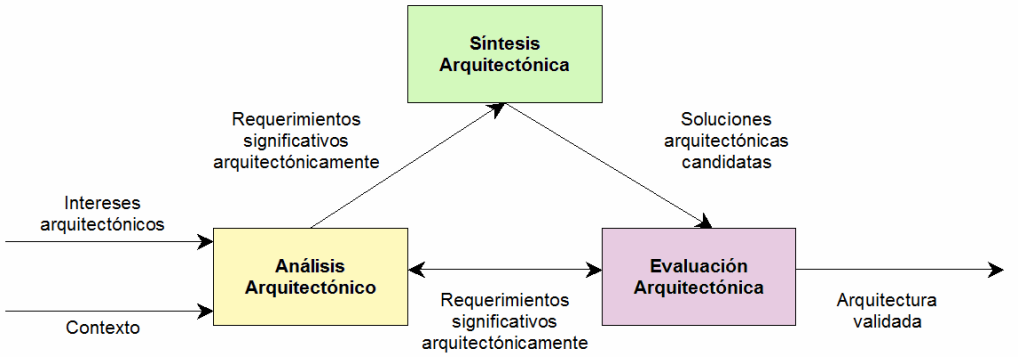

Figura. 2: Actividades del diseño arquitectónico, extraído de [5].

Debido a que no es posible analizar, encontrar soluciones y evaluar la arquitectura para todos los intereses de todos los "stakeholders" de manera simultánea, es que se divide el problema y se comienza trabajando sobre una parte del mismo, según los arquitectos lo consideren conveniente. Se lleva a cabo una secuencia iterativa e incremental de actividades de análisis-síntesis-evaluación. En cada iteración el arquitecto puede proponer y aplicar una o mas soluciones, las cuales pueden ser soluciones nuevas e innovadoras o bien soluciones ya conocidas que son adaptadas al nuevo problema para su reutilización. Ambas son el resultado de actividades altamente cognitivas por parte del arquitecto.

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 [6]. 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 ([7], [8]) 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 en el marco de esta Tesis [9], se pudo observar que el 98.5% de los encuestados hacía uso de la reutilización durante el diseño arquitectónico. Raras veces los arquitectos comienzan a trabajar a partir de una hoja en blanco ([10], [11]).

2. Identificación del problema

En la actualidad, dentro del contexto del diseño arquitectónico existen varias herramientas que brindan asistencia a los arquitectos para la realización de sus tareas. Algunas permiten modelar los elementos de las descripciones arquitectónicas (entre las que se incluyen los ADLs siendo UML el más frecuentemente utilizado) y otras además brindan la posibilidad de documentar y gestionar el conocimiento arquitectónico subyacente a los diseños creados.

Estas herramientas permiten que el arquitecto capture elementos pertenecientes a la arquitectura diseñada y pueda recuperarlos y visualizarlos posteriormente para su evolución o evaluación. Sin embargo, a pesar del importante papel que juega la reutilización en el diseño de una arquitectura, estas herramientas no tienen un mecanismo o procedimiento definido que le brinde asistencia al arquitecto en la realización y sistematización de dicha práctica. Es decir, el arquitecto no cuenta con un soporte que le facilite la realización del razonamiento analógico que se lleva a cabo durante la reutilización de un diseño arquitectónico. No ocurre lo mismo en etapas de diseño más avanzadas en donde se detalla la forma en la que la arquitectura de un sistema de software será implementada. Ejemplos de esto se describen en [12] y [13].

3. Objetivos de la Tesis

El objetivo general de la Tesis está directamente vinculado a brindar soporte al arquitecto de software durante el diseño arquitectónico. Partiendo de la hipótesis de que los arquitectos de software reutilizan soluciones conocidas en nuevos diseños, se identifica la necesidad de contar con una herramienta, tanto conceptual como computacional, que pueda colaborar con los arquitectos de software en dos aspectos:

- uno individual, que se encuentra relacionado con facilitar a los arquitectos la recuperación de experiencias pasadas propias, haciéndolas accesibles de manera de que puedan mapearlas con los problemas actuales para dar solución al diseño de un sistema;
- uno grupal, vinculado con facilitar a los arquitectos la recuperación de experiencias pasadas de otros arquitectos, haciéndolas accesibles de manera de que puedan mapearlas con los problemas actuales para dar solución al diseño de un sistema.

4. La propuesta

Para alcanzar el objetivo mencionado, se propone la aplicación de la técnica de Razonamiento Basado en Casos en el ámbito del diseño arquitectónico. Razonamiento Basado en Casos ha sido caracterizado como una instancia particular del razonamiento analógico en el cual se dan analogías intra-dominios, en lugar de inter-dominios [14]. Difiere de otros enfoques de Inteligencia Artificial debido a que enfatiza el rol de la memoria en el proceso cognitivo del ser humano. Su aplicación favorece el aprendizaje por experiencia, dado que generalmente es más sencillo aprender reteniendo la experiencia de resolver un problema concreto que obtener generalizaciones a partir de un problema [15].

La utilidad esperada de la propuesta presentada involucra: (i) la mejora en la calidad del producto de software resultante, y (ii) la reducción de los costos de los proyectos de software debido a la disminución del esfuerzo empleado por los arquitectos para diseñar las arquitecturas correspondientes.

Para un único individuo, es difícil manejar mentalmente la complejidad de la totalidad de los sistemas actuales cuando estos son grandes y deben satisfacer muchos requerimientos y restricciones. Al contar con una herramienta que le facilite el recuerdo de experiencias pasadas similares, el arquitecto podrá:

- acceder a información de soluciones de diseños arquitectónicos llevados a cabo en el pasado y analizar las decisiones tomadas para determinar si pueden ser replicadas en el sistema actual. Esta información puede provenir de experiencias propias o de otros arquitectos nutriendo el conjunto de posibles soluciones a emplear,
- evitar la "vaporización el conocimiento", y recordar sistemas o soluciones que por algún motivo pueden ser olvidadas o confundidas,
- contar con información proveniente de la experiencia acerca de cuales fueron las mejores soluciones aplicadas en el pasado, y cuales no deberían ser replicadas por haber considerado que su aplicación no fue adecuada.

Estos tres puntos describen el impacto en el esfuerzo llevado a cabo por los arquitectos durante el diseño arquitectónico, potenciando el tercero la disminución del esfuerzo por "retrabajo" al colaborar para que no se apliquen soluciones clasificadas como erróneas en situaciones anteriores que son consideradas similares a las nuevas. Además, el poder contar con información relacionada a la calidad de los sistemas diseñados con anterioridad propicia a mejorar la calidad resultante de los productos de software diseñados.

Como se pudo concluir a partir del análisis del trabajo de Tang y otros [16] y de la encuesta realizada en el marco de esta Tesis, durante un diseño arquitectónico, uno de los recursos más limitados para los arquitectos es el tiempo, por lo tanto para que la herramienta sea de utilidad:

- deberá ser fácil de usar para no entorpecer el trabajo de los arquitectos, y
- no deberá 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.

5. Descripción de la solución

Razonamiento Basado en Casos es un paradigma de resolución de problemas [15] que involucra el uso de experiencias pasadas para comprender y resolver nuevas situaciones [17]. 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 [15]. Suele estar compuesto por [18]:

- el problema: que describe el estado del mundo cuando ocurre el caso,
- la solución: que establece la solución encontrada, y/o
- la salida: que describe el estado del mundo luego de que ocurrió el caso.

Esta técnica involucra la ejecución de cuatro actividades conocidas como las *4Res* (Recuperar, Reusar, Revisar y Retener), las cuales son presentadas en la Fig. 3.

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. Generalmente es llevada a cabo en dos pasos [17]:

- primero se recupera un conjunto de casos candidatos. Los casos candidatos son casos con potencial para que a partir de ellos se efectúen predicciones relevantes sobre el nuevo caso,
- luego, se realiza un procesamiento para seleccionar el o los mejores casos del conjunto de casos candidatos previamente recuperados.

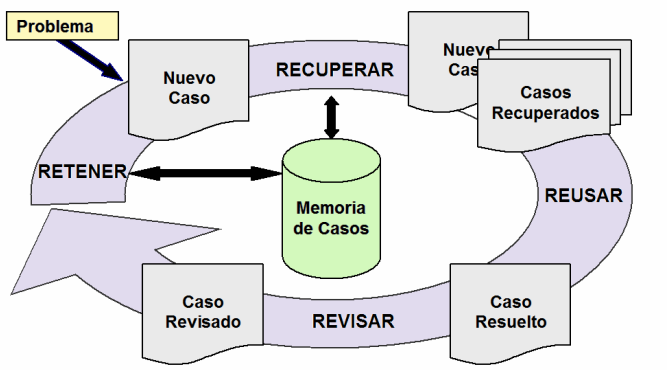

Figura 3: Actividades del proceso de Razonamiento Basado en Casos. Figura adaptada de [15].

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 [17].

La tercera actividad es la de **REVISIÓN** de la solución propuesta. Una vez que el nuevo caso ha sido resuelto debe ser probado en el mundo real. 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 [15]: la evaluación del caso resuelto generado por la actividad de reuso, y la reparación del caso resuelto, si es necesario, utilizando conocimiento específico del dominio.

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

Es importante aclarar que las actividades de reuso, revisión y retención raras veces ocurren sin la intervención humana [18].

5.1. Representación de casos en el contexto del diseño arquitectónico

Un razonador basado en casos es fuertemente dependiente de la estructura y el contenido de sus colecciones de casos [15]. En el contexto de esta Tesis, un caso describe la arquitectura diseñada para un sistema de software en particular y es llamado *caso arquitectónico*. A diferencia de la mayoría de las aplicaciones de Razonamiento Basado en Casos, en esta Tesis se utiliza un enfoque estructural [19] en donde los casos no se encuentran organizados como pares objeto-valor, sino que han sido definidos con un modelo de objetos complejo. Este modelo es representado utilizando diagramas de clases de UML ([20]) y es enriquecido mediante el empleo de OCL ([21]) para expresar restricciones definidas sobre el mismo.

Como puede observarse en la Fig. 2, cuando se debe diseñar la arquitectura de un sistema de software, la mayoría de la información con la que se cuenta está dada por los intereses arquitectónicos de los "stakeholders" y el contexto, o entorno, en el cual se inserta el sistema (ya sea operacional, político, organizacional, de negocio, etc.). Ambos conceptos ayudan a describir el problema asociado a un caso arquitectónico en particular. Por otro lado, una vez que el caso arquitectónico ocurre, es decir, luego de que la arquitectura del sistema ha sido diseñada, se obtiene como salida una arquitectura validada compuesta de elementos arquitectónicos junto con el razonamiento asociado a las decisiones de diseño tomadas por los arquitectos.

En esta Tesis, los casos arquitectónicos han sido estructurados en base al estándar internacional ISO/IEC/IEEE 42010:2011 [22] que define un marco conceptual estableciendo los términos y conceptos relacionados con el contenido y el uso de las descripciones arquitectónicas.

Dada la generalidad de los conceptos involucrados en el estándar ISO/IEC/IEEE 42010:2011, algunos de ellos han sido especializados para poder cumplir con el objetivo planteado previamente.

De esta forma, un caso arquitectónico se compone de un problema y una solución.

El *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.

En una etapa temprana de la definición de una arquitectura de software, los requerimientos más relevantes son aquellos asociados con la calidad del sistema de software a diseñar, los cuales son conocidos como requerimientos de calidad. Un requerimiento de calidad describe alguna característica o atributo de calidad que la solución de software debe poseer, como ser tiempo de respuesta rápido, facilidad de uso o alta confiabilidad [23].

Debido a los problemas de ambigüedades, inexactitudes e inconsistencias [24] que presenta el lenguaje natural, éste no puede ser empleado como herramienta para especificar los requerimientos en el contexto de esta Tesis. En cambio, se propone la utilización de *escenarios de atributos de calidad* (adoptados del trabajo de Bass y otros [1]) 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.

Los escenarios generales describen cómo una arquitectura permitiría responder a ciertos estímulos representando atributos de calidad [25]. Son genéricos e independientes del sistema y proveen un “framework” para formar un gran número de escenarios específicos del sistema [1]. Debido a su generalidad, pueden ser utilizados para crear los escenarios específicos de todos los sistemas diseñados. Los escenarios específicos del sistema expresan los requerimientos de calidad.

La representación empleada de escenarios fue complementada con la especificación de restricciones de diseño. Las *restricciones de diseño* son limitaciones sobre el diseño arquitectónico establecidas por los “stakeholders” del sistema. Estas restricciones acotan la libertad de los arquitectos al momento de tomar decisiones, imponiendo o impidiendo la utilización de determinadas estrategias de diseño, como ser tácticas o estilos arquitectónicos.

Los arquitectos utilizan estilos y tácticas arquitectónicas como ayuda en el proceso de diseño. Ambos están destinados a mejorar las decisiones de software de arquitectura y simplificar el proceso de diseño [26]. Las tácticas arquitectónicas están focalizadas en los atributos de calidad, son decisiones de diseño que ayudan a lograr una respuesta específica de un atributo de calidad en particular [27] por medio de la manipulación de algunos aspectos del modelo de atributo de calidad. Los estilos arquitectónicos son soluciones bien conocidas a problemas comunes de diseño [1]. Un estilo arquitectónico es una especialización de tipos de elementos y relaciones junto con un conjunto de restricciones sobre cómo pueden ser utilizados [2].

En principio, las restricciones de diseño limitan el espacio de soluciones disponibles. Frente a un conjunto de restricciones establecidas, es responsabilidad del arquitecto de software analizar y decidir si puede construir un sistema que las satisfaga. Para ello, es útil que cuente con información adicional como ser el grado de necesidad del cumplimiento de las restricciones (si son obligatorias u opcionales). Esta información cobra relevancia cuando las estrategias a aplicar para cumplir dos o más restricciones de diseño presentan conflictos entre si.

En la *solución* de un caso arquitectónico se describe la resolución del problema en términos que faciliten su reutilización en nuevos casos. Para determinar los elementos del diseño arquitectónico a reutilizar, es necesario tener en cuenta que existen dos niveles de reuso posibles [28]: (i) el reuso de las ideas y el conocimiento, y (ii) el reuso de los artefactos de diseño particulares y componentes. Debido a que la solución propuesta a los arquitectos debe ser de utilidad para el diseño de la nueva arquitectura, en el contexto de esta Tesis, la reutilización se lleva a cabo en el primer nivel: el del conocimiento. Los motivos que justifican esta elección están relacionados con dos hechos:

- Por un lado, algunas veces la correcta identificación de los artefactos que conforman el diseño arquitectónico depende del dominio específico de la aplicación. Teniendo en

cuenta esto, brindar a los arquitectos una solución para el nuevo caso basada en los artefactos y componentes de un caso recuperado de la memoria implicaría encontrar correspondencias entre los dominios de ambos casos. Debido a que uno de los requerimientos asociados a la propuesta presentada en esta Tesis, establece que no se debería requerir mayor esfuerzo por parte del arquitecto del que lleva documentar correcta y completamente una arquitectura, obligar a los arquitectos a documentar las arquitecturas con el suficiente nivel de detalle que permita establecer correspondencias entre dos dominios de aplicaciones aleatorios, violaría dicho requerimiento. Por lo tanto, para definir la estructura de la solución de un caso arquitectónico, se considera suficiente que los arquitectos puedan conocer cuales fueron las soluciones planteadas en los casos recuperados de manera genérica.

- Por el otro lado, contar únicamente con la identificación de los artefactos y componentes puede resultar inútil si no se acompaña de información vinculada a su uso. Por ejemplo, que el arquitecto del nuevo sistema conozca que para satisfacer un requerimiento de modificabilidad se utilizó un componente que representa al mapeador objeto-relacional "Hibernate" no es un aporte significativo. En cambio, es útil que el arquitecto del nuevo sistema conozca que para satisfacer un requerimiento de modificabilidad se decidió aplicar una táctica de separación de responsabilidades, motivo por el cual se utilizó un componente que representa al mapeador objeto-relacional "Hibernate".

Desde esta perspectiva, los conceptos seleccionados para llevar a cabo la reutilización del conocimiento de los arquitectos son los de decisiones de diseño arquitectónico y estrategias de diseño. Ambos conceptos forman parte de la solución de un caso arquitectónico. Las decisiones de diseño arquitectónico son los elementos que establecen cuales fueron los objetivos del arquitecto durante la construcción de las descripciones arquitectónicas y de que manera cumplieron dichos objetivos.

5.2. Recuperación de las decisiones de diseño

La aplicación de la técnica de Razonamiento Basado en Casos para un caso en particular comienza con la recuperación desde la memoria de los casos más similares. En la presente Tesis, esta actividad involucra la realización de un proceso complejo, esquematizado en la Fig. 4.

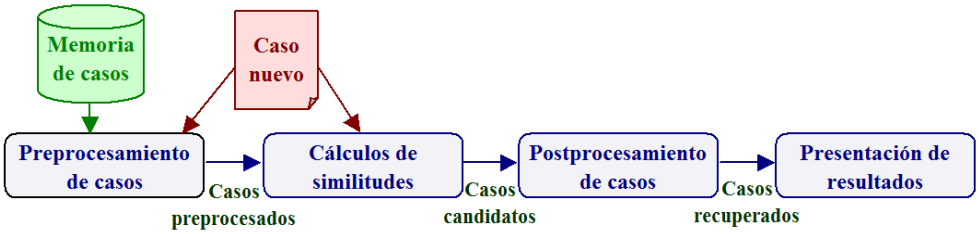

Figura 4: Etapas involucradas en la recuperación de casos arquitectónicos

Durante la primera etapa: **Preprocesamiento de casos** se recorren todos los *casos almacenados en la memoria* analizándolos desde la perspectiva de las restricciones de diseño impuestas por los "stakeholders" en el nuevo caso. Como resultado de esta etapa se obtiene un conjunto de *casos preprocesados*, los cuales son generados a partir de aquellos casos de la memoria que satisfacen las restricciones de diseño obligatorias del nuevo caso. La segunda etapa: **Cálculo de similitudes** está enfocada en los requerimientos de calidad solicitados por los "stakeholders" (documentados mediante escenarios específicos del sistema): se itera sobre todos los casos preprocesados obtenidos en la etapa anterior y se calcula la similitud entre cada uno de ellos y el nuevo caso analizando la similitud que existe entre los escenarios específicos del sistema de ambos casos (nuevo y candidato). Como resultado de esta etapa se obtiene un conjunto de *casos*

candidatos. En la tercera etapa: **Postprocesamiento de casos** se recorren todos los casos candidatos con el objetivo de analizar las decisiones documentadas en ellos para determinar cuán recomendables son las estrategias de diseño que serían propuestas a los arquitectos de software para dar solución al caso nuevo. Los casos candidatos se transforman así en *casos recuperados*. Finalmente, en la cuarta etapa: **Presentación de resultados** se exhiben los resultados obtenidos recordando y recomendando, a partir de cada caso recuperado, estrategias de diseño a los arquitectos de software que participan del diseño de la arquitectura del nuevo caso.

5.3. Reuso, revisión y retención

Como resultado de la actividad de recuperación, los casos obtenidos de la memoria son informados a los arquitectos de software involucrados en el diseño arquitectónico del nuevo sistema. A continuación, y como parte de la técnica de Razonamiento Basado en Casos propuesta, los arquitectos deben analizar las soluciones presentadas y seleccionar una para su *reuso*. La solución seleccionada se constituye como la base para la definición de la nueva arquitectura, no es en si misma una arquitectura completamente definida, por lo que los arquitectos deberían trabajar sobre ella. Cabe destacar que con la aplicación de la técnica de Razonamiento Basado en Casos propuesta en esta Tesis, no se intenta desplazar o reemplazar al arquitecto ni automatizar por completo su trabajo, sino que el objetivo es facilitar su labor y la de todos aquellos que comparten la memoria de casos, recordándoles experiencias pasadas que pueden ser de utilidad para la construcción del nuevo sistema.

Para que el caso que describe una arquitectura sea retenido en la memoria de casos debe ser previamente *revisado* y, si se detectan errores o problemas, debe ser corregido. Las evaluaciones y verificaciones deben ser llevadas a cabo desde dos puntos de vista:

- desde el punto de vista de diseño ingenieril, efectuando las acciones necesarias para garantizar que la arquitectura diseñada permite satisfacer los intereses de los "stakeholders", y
- desde el punto de vista de la técnica propuesta, efectuando validaciones para determinar si la arquitectura documentada cumple con los requisitos necesarios para poder constituirse como un caso arquitectónico disponible para ser utilizado como parte de la memoria de casos en un nuevo ciclo de la técnica de Razonamiento Basado en Casos

Finalizada la revisión, el caso puede ser *retenido* en la memoria de casos. De esta manera, estará disponible ante la llegada de un nuevo caso que implique una nueva ejecución del ciclo de Razonamiento Basado en Casos.

6. Utilización y aplicación de la propuesta

La propuesta fue evaluada y analizada desde dos puntos de vista: uno funcional y otro de calidad. Desde el punto de vista funcional, se trabajó sobre un caso de estudio referido al diseño de la arquitectura de un sistema de software en particular con el objetivo de materializar la propuesta en el contexto de casos reales extraídos de la industria. Surge, como producto de la observación del caso de estudio, la demostración de que la propuesta no necesariamente requiere una gran inversión de tiempo por parte de los arquitectos, ya que toda la información obtenida de la etapa de análisis de un ciclo de vida de desarrollo de software (restricciones de diseño y escenarios específicos del sistema) puede ser documentada por los analistas de software. De esta forma, los arquitectos sólo deben volcar en el caso arquitectónico la información que describe a las decisiones de diseño tomadas.

Desde el punto de vista de calidad, se analizó la aplicación de la técnica propuesta desde uno de los aspectos de calidad de software: "performance". ¿Cuánto tiempo debe esperar el arquitecto de software para contar con la propuesta de estrategias a aplicar una vez que ha iniciado la consulta?. Para analizar el tiempo de procesamiento necesario para dar respuesta a una consulta se llevaron a cabo pruebas de performance con distintos lotes de datos. Para ejecutar estas pruebas se

implementó una aplicación que trabajaba con datos aleatorios y permitía efectuar las mediciones necesarias para conocer el tiempo de procesamiento, tomando como parámetros de entrada la cantidad de casos, la cantidad de escenarios específicos del sistema y la cantidad de restricciones.

Los parámetros de entrada fueron establecidos de manera de poder combinarlos para ejecutar varias pruebas considerando una cantidad de casos en el rango de [20, 200], de escenarios específicos del sistema en el rango de [20, 500] y restricciones de diseño en el rango de [5 y 75].

Como resultado del estudio llevado a cabo se pudo concluir que si bien el tiempo promedio de espera aumenta en casos con muchos escenarios específicos del sistema, o cuando la memoria de casos es muy grande, el valor obtenido está dentro de un rango de valores razonable que no impactaría en el trabajo diario de los arquitectos.

También se trabajó en una herramienta, llamada RADS, que es un prototipo desarrollado para ofrecer a los arquitectos un soporte computacional que formaliza y materializa la propuesta realizada en esta Tesis. Dicha herramienta, implementada en lenguaje JAVA como un complemento de Eclipse, permite la captura de escenarios generales, estrategias de diseño, de los intereses planteados por los "stakeholders" de un sistema, y las decisiones tomadas por los arquitectos de software. Ofreciendo además la posibilidad de recuperar las estrategias de diseño aplicadas en casos previos para emplearlas en nuevos sistemas.

7. Aportes de la Tesis.

Los principales aportes de la Tesis se describen a continuación:

- Definición de un modelo que identifica los conceptos necesarios para representar arquitecturas de software como casos en el contexto de una técnica de Razonamiento Basado en Casos.
- Definición de una metodología que especifica las etapas necesarias para poder aplicar la técnica de inteligencia artificial de Razonamiento Basado en Casos a un problema de ingeniería de software como lo es el diseño arquitectónico.
- Aplicación de la técnica de Razonamiento Basado en Casos utilizando casos modelados mediante el paradigma orientado a objetos cuya estructura es compleja.
- Definición de una estrategia de recuperación compleja.
- Definición de una medida de similitud, desde la perspectiva de requerimientos no funcionales, que permite comparar dos casos arquitectónicos, y por consiguiente, dos arquitecturas de software que hayan sido representadas mediante casos.
- Organización de la información relacionada a las arquitecturas diseñadas en una memoria de casos compartida y de libre acceso que facilita la cooperación entre los arquitectos de software e impulsa la colaboración al establecer un repositorio común para capturar experiencias de diseño.

8. Futuras líneas de investigación.

Los trabajos futuros a partir de esta propuesta se enmarcan en cuatro grandes líneas, las cuales se describen a continuación:

- Seguimiento de las decisiones de diseño: Se considera, en esta línea de investigación, la utilidad de contar con información relacionada con la utilización de las estrategias de diseño reutilizadas como producto del empleo de la propuesta presentada en esta tesis.
- Enriquecimiento de la información suministrada acerca de los estilos arquitectónicos: Esta línea de investigación está impulsada por la necesidad de brindar al arquitecto de software información útil y válida que le permita tomar mejores decisiones durante el diseño arquitectónico.
- Migración de diseños arquitectónicos existentes: En el momento en que una organización decide utilizar la propuesta presentada en esta Tesis es probable que sus arquitectos ya hayan diseñado varias arquitecturas. También es probable que la descripción de dichas arquitecturas y la documentación del razonamiento subyacente a ellas esté en distintos medios, digitales o físicos, como ser archivos de texto, diagramas UML, notas, correos electrónicos, etc. La incorporación a la memoria de casos de las experiencias de diseño arquitectónico vividas previamente permite

comenzar a utilizar la propuesta contando con casos disponibles plausibles de ser recuperados ante una consulta puntual.

- Perfeccionamiento de la herramienta RADS: Esta línea de trabajo tiene como objetivo poder llevar a cabo las modificaciones e implementaciones necesarias para la integración de RADS con otras herramientas de modelado UML empleadas por los arquitectos de software de la industria.

9. Referencias

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, Second Edition.* Addison-Wesley, 2003.

[2] 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

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

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

[5] Ch. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, 2007.

[6] 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.

[7] D. Gentner. Similarity and analogical reasoning. chapter The Mechanisms of Analogical Learning, pages 197–241. Cambridge University Press, 1989.

[8] 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.

[9] 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.

[10] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, Third Edition.* Addison-Wesley, 2013.

[11] P. Eeles and P. Cripps. *The Process of Software Architecting*. Pearson Education, 2009.

[12] G. Vazquez, J. A. Diaz Pace, and M. Campo. Reusing design experiences to materialize software architectures into object-oriented designs. *Information Sciences*, 259:396–411, February 2014.

[13] P. Gomes and A. Leitäo. A tool for management and reuse of software design knowledge. In Steffen Staab and Vojtech Svátek, editors, *Managing Knowledge in a World of Networks*, volume 4248 of *Lecture Notes in Computer Science*, pages 381–388. Springer Berlin / Heidelberg, 2006.

[14] B. López. *Case-Based Reasoning: A Concise Introduction.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[15] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications. IOS Press*, 7(1):39–59, 1994.

[16] A. Tang, M. Ali Babar, I. Gorton, and J. Han. A survey of architecture design rationale. *The Journal of Systems and Software, Elseiver*, 79:1792–1804, 2006.

[17] J. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6:3–34, 1992.

[18] I. Watson and F. Marir. Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):327–354, 1994.

[19] R. Bergmann, K-D. Althoff, S. Breen, M. Göker, M. Manago, R. Traphöner, and S. Wess. Case-based reasoning approaches. In *Developing Industrial Case-Based Reasoning Applications*, volume 1612 of *Lecture Notes in Computer Science*, pages 21–34. Springer Berlin Heidelberg, 2003.

[21] OMG. Object constraint language, version 2.2. http://www.omg.org/spec/OCL/2.2, 2010.

[20] OMG. Unified modeling language, version 2.3. http://www.omg.org/spec/UML/2.3, 2010.

[22] International Organization for Standardization and International Electrotechnical Commission. Systems and software engineering - architecture description (ISO/IEC/IEEE 42010). edition 01-12-2011. 2011.

[23] S. Pfleeger and J. Atlee. *Software Engineering: Theory and Practice*. Prentice Hall, 2009.

[24] L. Bass, J Berget, P. Clements, P Merson, I Ozkaya, and R Sangwan. A comparison of requirements specification methods from a software architecture perspective. Technical Report Tech. Rep. CMU/SEI-2006-TR-013, SEI, 2006.

[25] L. Bass, M. Klein, and G. Moreno. Applicability of general scenarios to the architecture tradeoff analysis method. Technical Report CMU/SEI-2001-TR-014, SEI, 2001.

[26] F. Bachmann, L. Bass, and R. Nord. Modifiability tactics. Technical Report CMU/SEI-2007-TR-002, SEI, 2007.

[27] F. Bachmann, L. Bass, and M. Klein. Illuminating the fundamental contributors to software architecture quality. Technical Report CMU/SEI-2002-TR-025, SEI, 2002.

[28] R. Prietro-Díaz and P. Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6, 1987.