# Reutilización de Soluciones Arquitectónicas empleando Razonamiento Basado en Casos

María Celeste Carignano[1,2] , Silvio Gonnet[1,2,3], Horacio Leone[1,2,3]
[1] INGAR, Instituto de Desarrollo y Diseño
[2] CIDISI, Universidad Tecnológica Nacional – Facultad Regional Santa Fe
[3] CONICET, Comisión Nacional de Investigaciones Científicas y Técnicas
{celestec, sgonnet, hleone}@santafe-conicet.gov.ar

**Resumen.** El diseño de arquitecturas de software es una actividad creativa y compleja realizada principalmente por arquitectos de software. Consiste de una colección de decisiones que generalmente están influenciadas por la experiencia y los conocimientos de los arquitectos involucrados. El razonamiento basado en casos es una técnica de inteligencia artificial que se emplea en este trabajo para brindar soporte al diseño, recordando a los diseñadores experiencias previas que pueden ayudar en nuevas situaciones, mediante la reutilización de soluciones arquitectónicas empleadas con anterioridad. El objetivo principal es proveer al arquitecto de un conjunto de soluciones que podrían ser aplicadas como una propuesta inicial para diseñar la arquitectura del sistema, que luego deberá ir extendiendo hasta incluir todos los requerimientos. Se describe también una herramienta desarrollada para dar soporte a la técnica planteada y un caso de estudio en dónde queda plasmada la propuesta.

**Palabras claves:** diseño de arquitecturas de software, razonamiento basado en casos, reutilización.

## 1 Introducción

La reutilización es el uso de conceptos u objetos previamente adquiridos en una nueva situación. Monroe y Garlan describen en [1] varios beneficios asociados con la reutilización del diseño de software, también aplicables al diseño de arquitecturas de software. Primero, dado que las decisiones de diseño dirigen las primeras fases del desarrollo de un sistema, muchos de los errores iniciales (a menudo los más costosos) pueden ser evitados. Segundo, la reutilización de un diseño familiar puede mejorar la comprensibilidad de un sistema, facilitando la evolución y el mantenimiento del mismo. Tercero, la reutilización del diseño promueve la reutilización de código: a menudo gran parte de la infraestructura que da soporte al diseño puede ser compartida entre las aplicaciones que comparten tal diseño.

El diseño de arquitecturas de software es una actividad creativa y compleja realizada principalmente por arquitectos de software. Dado un conjunto de requerimientos solicitados por los "stakeholders", tanto funcionales como de calidad, el arquitecto decide cuáles serán los elementos arquitectónicos más apropiados para satisfacerlos. Un diseño arquitectónico consiste de una colección de decisiones [2], las

cuales generalmente están influenciadas por la experiencia de los arquitectos [3] y están basadas en la aplicación de soluciones bien conocidas o una combinación de elementos realizada previamente cuyo impacto en el sistema resultante fue bueno.

En esta contribución se presenta una aplicación de la técnica de razonamiento basado en casos (RBC) para dar soporte a la reutilización de soluciones arquitectónicas, de manera que éstas puedan ser representadas y almacenadas en una base de casos (memoria) para posteriormente ser recuperadas y adaptadas. El objetivo principal es proveer al arquitecto de un conjunto de soluciones para ser empleadas inicialmente al diseñar la arquitectura del sistema, que luego deberá ir extendiendo para incluir todos los requerimientos.

Las soluciones propuestas al arquitecto serán un conjunto de estilos arquitectónicos y tácticas arquitectónicas, junto con soluciones particulares previamente aplicadas por otros arquitectos. Un estilo arquitectónico determina el vocabulario de elementos arquitectónicos que pueden ser utilizados en instancia de ese estilo, junto con un conjunto de restricciones sobre cómo pueden ser combinados [2]. Una táctica arquitectónica es una decisión de diseño que permite lograr una respuesta de atributo de calidad deseada [4]. Ambos conceptos representan soluciones que pueden ser aplicadas para dar respuesta a objetivos particulares, como ser maximizar la performance de determinada funcionalidad, permitir encapsular cierta funcionalidad para evitar que los cambios afecten a otras, etc. La utilización de alguna de estas soluciones puede afectar a otras soluciones planteadas previamente, por ejemplo, algunas tácticas para lograr performance (como ser optimizar los algoritmos) pueden dificultar la aplicación de algunas tácticas para lograr modificabilidad (como ser dividir responsabilidades). Es por ello que en este trabajo se consideró conveniente permitir que el arquitecto cuente con un conjunto de soluciones aplicadas previamente para resolver un problema anterior similar, y no únicamente con una enumeración de tácticas o estilos arquitectónicos que se podrían aplicar para satisfacer cada requerimiento de manera aislada.

Se propone que el RBC de soporte al diseño recordando a los diseñadores experiencias previas que puedan guardar semejanzas con las nuevas situaciones [5]. RBC es un paradigma de resolución de problemas que involucra utilizar experiencias pasadas (casos pasados) para comprender y resolver nuevos problemas. Aamodt y Plaza plantean que un caso pasado generalmente denota una situación experimentada previamente, la cuál ha sido capturada y aprendida de una manera que pueda ser reutilizada en la solución de problemas futuros [6].

En las siguientes secciones se describe la propuesta y se brindan ejemplos de su aplicación. En la sección 2 se introduce la técnica de RBC propuesta. En la sección 3, se describe una herramienta desarrollada para dar soporte a la técnica planteada y algunos resultados de pruebas realizadas para evaluar la técnica. En la sección 4 se enumeran algunos trabajos relacionados, tanto desde el punto de vista de la arquitectura de software como del RBC. Finalmente, en la sección 5 se describen conclusiones y trabajos futuros.

## 2 Reutilizando soluciones arquitectónicas

La resolución de un problema utilizando RBC requiere de la realización de un conjunto de actividades, las cuales son enumeradas por López de Mántaras y Plaza [7] de la siguiente manera: i) obtener una descripción del problema, ii) medir la similitud del problema actual con los problemas almacenados previamente en la memoria, iii) *recuperar* uno o más casos similares e intentar *reutilizar* la solución de uno de los casos recuperados, iv) adaptar si es necesario para tener en cuenta las diferencias con el problema original, v) evaluar la solución propuesta, vi) si de la evaluación surge que es necesario, *revisar* la solución propuesta, vii) y finalmente *retener* la descripción del problema y su solución como un nuevo caso. Estas actividades pocas veces ocurren sin intervención humana [8].

A continuación se describen cada una de estas actividades en el marco de la reutilización de soluciones arquitectónicas creadas para diseñar un sistema de software.

### 2.1 Representación de casos

Un caso está conformado por dos partes principales: i) una descripción del problema y del entorno en el que el mismo ocurre, y ii) una descripción de la solución que fue aplicada para resolver el problema descrito.

Un razonador basado en casos es fuertemente dependiente de la estructura y contenido de sus casos [6]. Por lo tanto, a continuación se describe cómo se estructurará un caso en el contexto del desarrollo de un diseño arquitectónico. Los casos en esta contribución han sido definidos utilizando el paradigma de orientación a objetos y representados en diagramas de clase de UML 2.0.

La *descripción del problema* está dada por la identificación de los requerimientos de los "stakeholders" del sistema. Los requerimientos establecen de manera explícita qué funciones debe brindar el sistema y bajo qué parámetros de calidad. Aquellos requerimientos que están asociados a funcionalidades específicas que debe brindar el sistema se conocen como requerimientos funcionales (*FunctionalRequirement* en Figura 1). Por otro lado, los requerimientos que indican características que debe poseer el sistema (o alguna funcionalidad particular del mismo), como ser seguridad, tiempo de respuesta o confiabilidad, se conocen como requerimientos de calidad (*QualityRequirement* en Figura 1). Es responsabilidad del arquitecto diseñar una arquitectura de software que brinde las funcionalidades requeridas por los "stakeholders" cumpliendo los requerimientos de calidad solicitados.

La información relevada a través de requerimientos es información descriptiva (ya sea en lenguaje narrativo o en un lenguaje más formal, como ser UML) de lo que los "stakeholders" esperan de cada sistema en particular, por lo que no pueden compararse dos requerimientos de dos sistemas distintos a partir de sus descripciones. Sin embargo, se puede recurrir a los conceptos de escenario general de atributo calidad (*GeneralScenario* en Figura 1) y escenario concreto de atributo calidad (*ConcreteScenario* en Figura 1) para clasificar los requerimientos de calidad y poder efectuar comparaciones entre escenarios. Los escenarios generales proveen un "framework" para generar un gran número de escenarios específicos de atributos de

calidad genéricos e independientes del sistema, describiendo cómo la arquitectura debería responder a cierto estímulo [3].

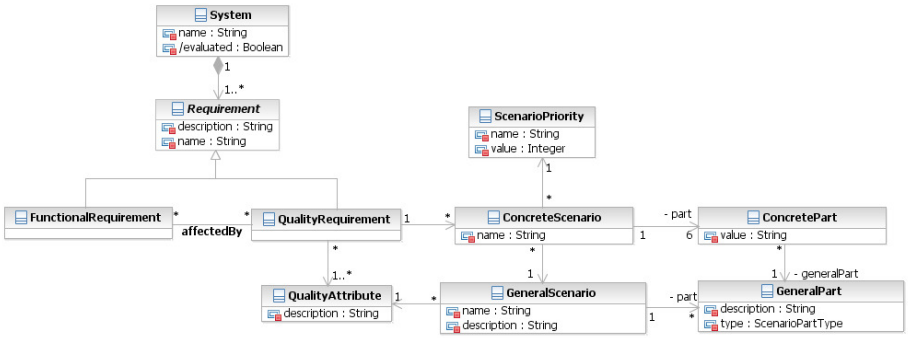


**Fig. 1.** Descripción del problema: identificación de requerimientos

En esta propuesta los escenarios generales permitirán establecer similitudes entre requerimientos de calidad debido a que son independientes del sistema. La asociación entre un escenario general y un requerimiento de calidad es realizada a través de escenarios concretos. De esta manera un escenario general puede ser visto como un "template" para generar uno o múltiples escenarios concretos [9].

Bass y otros describen en [3] que un escenario general de atributo de calidad se define a través de seis partes (*GeneralPart* en Figura 1): i) origen del estímulo: es la entidad (humana, sistema de información, u otra) que genera el estímulo; ii) estímulo: es la condición que debe ser considerada al arribar al sistema; iii) entorno: es la condición en que se encuentra el entorno al arribar el estímulo; iv) artefacto: parte del sistema que es estimulada; v) respuesta: es la actividad que se realiza después de la llegada del estímulo; vi) medida de la respuesta: es la medida que se considera para verificar que el requisito se cumple luego de producida la respuesta. Un ejemplo de escenario general del atributo de calidad modificabilidad es presentado en la Tabla 1.

**Tabla 1**. Ejemplo de escenario general del atributo de calidad modificabilidad [3]

| Parte del escenario | Posibles valores |
|---|---|
| Origen del estímulo | Usuario final, desarrollador, administrador del sistema |
| Estímulo | Deseos de:<br>- agregar, modificar, eliminar una funcionalidad<br>- agregar, modificar, eliminar una capacidad<br>- agregar, modificar eliminar un atributo de calidad |
| Artefacto | Interfaz de usuario, Plataforma, Entorno, Sistemas que operan con el sistema a diseñar |
| Entorno | En tiempo de ejecución, de compilación, de diseño, de desarrollo |
| Respuesta | - Localizar los lugares a ser modificados en la arquitectura<br>- Hacer las modificaciones sin afectar otras funcionalidades<br>- Probar las modificaciones<br>- Distribuir las modificaciones |
| Medida de la respuesta | Costo en términos de elementos afectados o esfuerzo o dinero |

Un escenario concreto se describe indicando las partes del escenario general a partir del cuál se ha originado (a través de instancias de *ConcretePart* en Figura 1). Un ejemplo de escenario concreto es: "Un desarrollador desea cambiar la interfaz de usuario de un sistema. Este cambio deberá ser realizado en tiempo de diseño, deberá consumir menos de tres horas, tanto para el desarrollo como para las pruebas, y no deberán surgir efectos secundarios productos de su implementación".

A todos los escenarios concretos se les asigna una prioridad, la cuál indica la importancia que tiene el alcance de dicho escenario para el sistema completo. La arquitectura de un sistema no es la misma cuando se consideran como prioritarios requerimientos de performance, que cuando se consideran como prioritarios requerimientos de modificabilidad.

Como se indicó al comienzo de esta sección, un caso también describe la *solución* aplicada para resolver el problema identificado. Esta solución está conformada por todas las soluciones arquitectónicas aplicadas (*ArchitecturalSolutionApplied* en Figura 2) para poder satisfacer los escenarios concretos de atributos de calidad identificados. Es decir, se almacenan en los casos las decisiones de diseño arquitectónico (*ArchitecturalDesignDecision* en Figura 2) tomadas por los arquitectos, junto con información relevante asociada a las mismas, las que serán de utilidad al momento de comprender la solución obtenida.

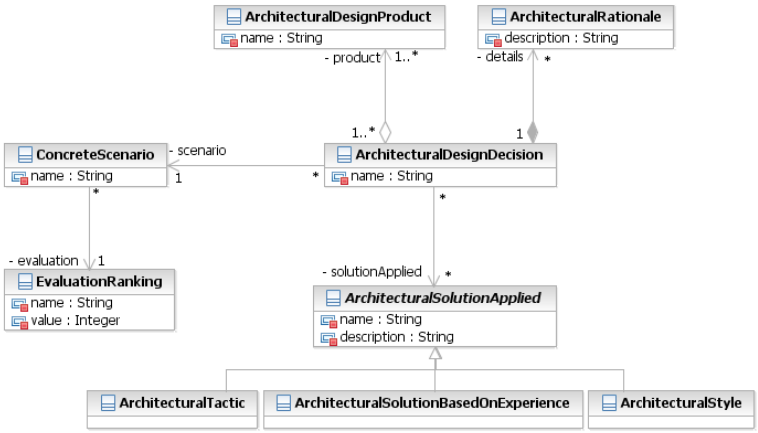


**Fig. 2.** Solución del caso.

La información adicional está compuesta por: i) el razonamiento realizado por los arquitectos que fundamente los motivos de la decisión (*ArchitecturalRationale* en Figura 2); ii) los productos del diseño arquitectónico (*ArchitecturalDesignProduct* en Figura 2) que describen el diseño arquitectónico final, el cual puede estar compuesto por módulos, componentes y conectores, u otros elementos arquitectónicos pertinentes.

Es importante poder contar con información asociada al grado de satisfacción que pudo alcanzar cada requerimiento de calidad al implementarse la arquitectura diseñada. Esto puede ser realizado asociando una calificación a cada escenario concreto (como instancias de *EvaluationRanking* en Figura 2) luego de implementado

el sistema, momento en el cuál puede evaluarse si se han satisfecho todos los requerimientos de calidad solicitados por los "stakeholders".

De esta manera queda definido el caso, el cuál registra la descripción del problema (dada por los requerimientos del usuario, tanto en su forma descriptiva como en el catálogo de escenarios), y la solución (dada por las soluciones arquitectónicas seleccionadas para satisfacer los escenarios y la información complementaria asociada).

Un caso será incorporado a la memoria una vez que haya podido evaluarse la satisfacción de los requerimientos de calidad a partir de la arquitectura generada. Esto sólo es posible luego de la implementación del sistema.

Es necesario cumplir dicho requisito ya que de esa manera se puede establecer el conocimiento necesario para poder discernir si la solución planteada a un escenario es útil o no, y de esta manera poder considerarlo al momento de recuperar casos para dar respuesta a uno nuevo.

### 2.2 Recuperación de casos

La tarea de recuperación de casos desde la memoria de casos comienza con una descripción del problema (caso nuevo), y finaliza con uno o varios casos de la memoria que poseen mayor similitud con el caso nuevo [6].

Para cada caso en la memoria (*casoM*) debe calcularse la similitud con el caso nuevo (*casoN*). La similitud es calculada comparando los escenarios concretos de atributos de calidad de ambos casos. El o los casos que presenten mayor similitud serán los seleccionados para continuar con la siguiente tarea de RBC.

Para resolver el problema de obtener el mejor valor de similitud entre dos casos se utiliza programación lineal entera binaria, ya que se debe calcular cuáles son los pares de escenarios concretos (uno de *casoM* y otro de *casoN*) que permiten lograr la máxima similitud total entre ambos casos (Ecuación 1). De esta manera se pueden identificar un subconjunto de escenarios concretos de *casoM* que son similares a un subconjunto de escenarios concretos de *casoN*.

$$max(\sum_{i=0}^{n}\sum_{j=0}^{m} similitud(escenario_i(casoN), escenario_j(casoM)) * x_{ij}))/n \qquad (1)$$

Sujeto a:

$$\forall i \sum_{j=1}^{m} x_{ij} <= 1 \qquad \forall j \sum_{i=1}^{n} x_{ij} <= 1 \qquad \forall i \forall j \;\; x_{ij} es\ binaria$$

Dónde:

n = cantidad de escenarios concretos del casoN

m = cantidad de escenarios concretos del casoM

Dados dos escenarios concretos: *escA* y *escB* dónde el primero representa al escenario concreto perteneciente a *casoN* y el segundo representa al escenario

concreto perteneciente a *casoM*, la similitud entre estos escenarios (Ecuación 2) es calculada en función de:

i) los escenarios generales (instancias de *GeneralScenario* en Figura 1) asociados a los escenarios concretos: se considera que si dos escenarios concretos no poseen el mismo escenario general no son escenarios similares (Ecuación 3),

ii) las prioridades (instancias de *ScenarioPriority* en Figura 1) definidas para ambos escenarios: se considera que las soluciones arquitectónicas aplicadas no serán las mismas cuando, por ejemplo, los requerimientos de performance son más prioritarios que los requerimientos de modificabilidad (Ecuación 4),

iii) las partes de cada escenario concreto (instancias de *ConcretePart* en Figura 1): con el objetivo de calcular la similitud con mayor precisión. Un escenario general define varios posibles valores para cada parte que lo compone (estímulo, origen del estímulo, artefacto, entorno, respuesta y medida de respuesta) y un escenario concreto selecciona un único valor por parte, por lo que se puede identificar en cuántas partes se asemejan dos escenarios concretos (Ecuaciones 5 y 6) comparando las partes de los escenarios generales que fueron seleccionadas,

iv) el resultado de la evaluación del *escB* (descrito en la asociación con una instancia de *EvaluationRanking* en Figura 2). Este valor es de suma importancia, y aplicarlo a la ecuación permite dar mayor importancia a aquellos escenarios que pudieron ser satisfechos de mejor manera con las soluciones que fueron seleccionadas para su aplicación.

Estos últimos tres elementos (ii, iii, y iv) son ponderados por un factor de peso: $w_i$ que permite determinar la importancia de cada uno en la función con el objetivo de fijar cuáles son los aspectos más prioritarios. Dependiendo el sistema, algunos aspectos serán más importantes que otros, o no.

$$\begin{aligned} similitud(escA, escB) \;=\; & similitud(escenarioGeneral(escA), escenarioGeneral(escB)) * \\ & (w1 * similitud(prioridad(escA), prioridad(escB)) + \\ & + w2 * similitud(partes(escA), partes(escB)) + \\ & + w3 * evaluacion(escB)) \end{aligned} \tag{2}$$

$$con\ w1 + w2 + w3 = 1$$

$$similitud(escGeneralA, escGeneralB) = \begin{cases} 1 & \text{si escGeneralA = escGeneralB;} \\ 0 & \text{en caso contrario.} \end{cases} \tag{3}$$

$$similitud(prio1, prio2) = (maxPrioridad - abs(prio1 - prio2))/maxPrioridad \tag{4}$$

$$\begin{aligned} similitud(partesConcA, partesConcB) \;=\; & (\sum_{i=1}^{6}\sum_{j=1}^{6} similitud(parteGeneral(partesConcA_i), \\ & parteGeneral(partesConcB_j)))/6 \end{aligned} \tag{5}$$

$$similitud(parteGeneralA, parteGeneralB) = \begin{cases} 1 & \text{si parteGeneralA = parteGeneralB;} \\ 0 & \text{en caso contrario.} \end{cases} \tag{6}$$

Las tareas de reutilización y revisión son tareas que requieren de la intervención del arquitecto y están ligadas muy estrechamente. El resultado de la tarea de recuperación será un conjunto de casos, con una similitud aceptable (definida como aquellos que superan un valor límite de similitud previamente fijado). Estos casos recuperados presentan las soluciones arquitectónicas aplicadas para satisfacer un subconjunto de escenarios concretos similares a un subconjunto de escenarios del caso nuevo.

Es responsabilidad del arquitecto decidir cuál será el caso a reutilizar y en qué medida se reutilizará, es decir, si se reutilizarán todas las soluciones o una parte de ellas, o si se reutilizará una combinación de soluciones brindadas por varios de los casos recuperados.

## 3 Implementación y evaluación

Se ha desarrollado una extensión de la herramienta Eclipse, llamada RADS (***R**euse **A**rchitectural **D**esign **S**olutions*), utilizando el lenguaje Java para implementar la técnica RBC propuesta. Las principales vistas que provee permiten crear sistemas, especificar los requerimientos funcionales y de calidad, describir los escenarios generales y concretos, documentar las decisiones de diseño (con una extensión al modelo propuesto para permitir generar múltiples soluciones arquitectónicas paralelas y múltiples decisiones de diseño para satisfacer un mismo escenario concreto) y aplicar la técnica propuesta de RBC para reutilizar soluciones arquitectónicas.

Se encuentra integrada con el plugin Rational Software Architect [10], que permite diseñar la arquitectura de forma gráfica, utilizando UML 2.0 y con la librería ILOG CPLEX [11] que permite realizar el cálculo de optimización para obtener los casos similares.

La evaluación de la propuesta se ha realizado en varias etapas: i) primero utilizando casos de estudio reales de menor tamaño, para verificar el funcionamiento con casos de la industria y ii) luego utilizando un conjunto de lotes de datos creados automáticamente (de distintos tamaños) para verificar que se obtuvieran resultados correctos en un tiempo acotado. A continuación se describe brevemente la primera etapa.

### 3.1 Evaluación de la propuesta utilizando casos reales.

Esta evaluación se realizó utilizando varios lotes de casos reales de la industria del software de pequeño tamaño, con el objetivo de permitir un mejor seguimiento de los resultados.

A continuación se describe una de las pruebas. Los casos considerados fueron tres. Los dos primeros se almacenaron en la memoria como casos aprendidos y el tercero fue utilizado para realizar la evaluación. Las soluciones de los casos almacenados en la memoria consistieron en la aplicación de las tácticas y los estilos arquitectónicos existentes en la literatura ([3] y [2] respectivamente).

El primer caso almacenado en la memoria corresponde a un sistema encargado de gestionar órdenes de servicio en un taller mecánico. En la tabla 2 se resumen los escenarios relevados y las soluciones aplicadas para satisfacerlos, a través de decisiones de diseño. La calificación igual a 75 % indica que las soluciones aplicadas para satisfacer dicho escenario fueron adecuadas pero no en su totalidad. La asignación de las soluciones arquitectónicas a los escenarios concretos se realiza a través de una o más decisiones de diseño (*ArchitecturalDesignDecision* en Figura 2).

**Tabla 2**. Escenarios de sistema de gestión de órdenes de servicio de taller mecánico

| Esc. | Breve Descripción | Atributo de Calidad | Prioridad | Califi-cación | Soluciones Arquitectónicas Utilizadas |
|---|---|---|---|---|---|
| **E01** | Evitar el ingreso de usuarios no autorizados bloqueando el acceso a la aplicación. | Seguridad | Alta | 75 % | Tácticas arquitectónicas:<br>- Autenticar usuarios |
| **E02** | Evitar que usuarios sin los permisos necesarios modifiquen datos de la aplicación. | Seguridad | Alta | 75 % | Tácticas arquitectónicas:<br>- Autorizar usuarios |
| **E03** | Permitir que se realicen modificaciones a la interfaz de usuario sin que ello impacte en otras funcionalidades del sistema. | Modificabilidad | Media alta | 75 % | Tácticas arquitectónicas:<br>- Ocultar información y utilizar encapsulamiento<br>- Mantener coherencia semántica<br><br>Estilos arquitectónicos:<br>- Modelo-Vista-Controlador |
| **E04** | Generar los reportes en menos de tres segundos. | Performance | Media alta | 75 % | Tácticas arquitectónicas:<br>- Incrementar la eficiencia computacional, reducir el "overhead" computacional |

El segundo caso almacenado en la memoria corresponde a un sistema de control de asistencia del personal de una empresa a través de tarjetas y lectores de huellas

digitales. En la tabla 3 se resumen los escenarios relevados y las soluciones aplicadas para satisfacerlos, a través de decisiones de diseño.

**Tabla 3**. Escenarios de sistema de control de asistencia de personal

| Esc. | Breve Descripción | Atributo de Calidad | Prioridad | Calificación | Soluciones Arquitectónicas Utilizadas |
|---|---|---|---|---|---|
| **E01** | Requerir como máximo tres días para configurar los dispositivos de autenticación. | Modificabilidad | Media alta | 75 % | Tácticas arquitectónicas:<br>- Anticipar cambios esperados<br>- Limitar posibles opciones<br>- Mantener coherencia semántica |
| **E02** | Mantener disponible el sistema en los períodos comprendidos entre las 7 y las 9 de la mañana y las 13 y 14 de la tarde. | Disponibilidad | Alta | 75 % | Tácticas arquitectónicas:<br>- Activar redundancia |
| **E03** | No permitir que el sistema se encuentre fuera de servicio por más de 5 minutos. | Disponibilidad | Alta | 75 % | Tácticas arquitectónicas:<br>- Activar redundancia<br>- Mantener múltiples copias de los datos<br>- Monitorear procesos<br>- Utilizar transacciones |
| **E04** | Permitir agregar nuevos formatos de reportes en no más de un día. | Modificabilidad | Media | 75 % | Tácticas arquitectónicas:<br>- Anticipar cambios esperados<br>- Mantener coherencia semántica<br><br>Estilos arquitectónicos:<br>- Modelo-vista-controlador |
| **E05** | Procesar la planilla de inasistencia de todos los empleados en menos de 3 segundos. | Performance | Media alta | 75 % | Tácticas arquitectónicas:<br>- Incrementar recursos disponibles<br>- Incrementar eficiencia computacional<br>- Reducir overhead computacional |
| **E06** | Registrar el ingreso o la salida de un empleado en menos de 3 segundos. | Performance | Media alta | 75 % | Tácticas arquitectónicas:<br>- Incrementar eficiencia computacional<br>- Reducir overhead computacional |

Finalmente, el caso utilizado para realizar la evaluación corresponde a un sistema para reproducir videos. En la tabla 4 se resumen los escenarios relevados.

**Tabla 4**. Escenarios de sistema de reproducción de videos

| Esc. | Breve Descripción | Atributo de Calidad | Prioridad |
|---|---|---|---|
| **E01** | Evitar ingreso de usuarios no autorizados bloqueando el acceso a la aplicación. | Seguridad | Alta |
| **E02** | Cargar la información contextual de la película en no más de 3 segundos | Performance | Media alta |
| **E03** | Cargar la película en no más de 3 segundos. | Performance | Media alta |
| **E04** | Comenzar a reproducir la película una vez recibido el requerimiento en no más de 1 segundo. | Performance | Media |
| **E05** | Evitar la degradación de la imagen por problemas de carga de la red. | Disponibilidad | Alta |
| **E06** | Restablecer el link con el servidor en no más de 5 segundos en casos de que ocurra alguna falla. | Disponibilidad | Alta |
| **E07** | No requerir de entrenamiento para que cualquier usuario pueda utilizar la aplicación. | Usabilidad | Media |

Las matrices de similitudes del sistema de reproducción de video (*casoNuevo*) con el sistema de gestión de órdenes de servicio (*casoMemOS*) y el sistema de control de asistencia del personal (*casoMemAP*) se presentan en las Tablas 5 a) y b), respectivamente. Los pesos asignados fueron: $w_1 = 0.30$, $w_2 = 0.30$, y $w_3 = 0.40$, otorgando mayor prioridad al grado de satisfacción obtenido para cada escenario de los casos en memoria.

**Tabla 5**. a) Matriz de similitud del sistema de reproducción de video y el sistema de gestión de órdenes de servicio. b) Matriz de similitud del sistema de reproducción de video y el sistema de control de asistencia del personal.

| **Sistema de reproducción de video** | **Sistema de gestión de ordenes de servicio** | | | | |
|---|---|---|---|---|---|
| | | **E01** | **E02** | **E03** | **E04** |
| | **E01** | 0.95 | 0.85 | 0 | 0 |
| | **E02** | 0 | 0 | 0 | 1 |
| | **E03** | 0 | 0 | 0 | 0.95 |
| | **E04** | 0 | 0 | 0 | 0.65 |
| | **E05** | 0 | 0 | 0 | 0 |
| | **E06** | 0 | 0 | 0 | 0 |
| | **E07** | 0 | 0 | 0 | 0 |

a)

| **Sistema de reproducción de video** | **Sistema de control de asistencia de personal** | | | | | | |
|---|---|---|---|---|---|---|---|
| | | **E01** | **E02** | **E03** | **E04** | **E05** | **E06** |
| | **E01** | 0 | 0 | 0 | 0 | 0 | 0 |
| | **E02** | 0 | 0 | 0 | 0 | 0.95 | 0.95 |
| | **E03** | 0 | 0 | 0 | 0 | 1 | 1 |
| | **E04** | 0 | 0 | 0 | 0 | 0.7 | 0.7 |
| | **E05** | 0 | 0.9 | 0.9 | 0 | 0 | 0 |
| | **E06** | 0 | 0.9 | 0.9 | 0 | 0 | 0 |
| | **E07** | 0 | 0 | 0 | 0 | 0 | 0 |

b)

Una vez calculada la similitud total entre ambos pares de sistemas, se obtienen los siguientes valores: 0.54 como medida de similitud entre *casoNuevo* y *casoMemOS* y 0.28 como medida de similitud entre *casoNuevo* y *casoMemAP*. En la Figura 3, puede observarse esta información en una captura de pantalla de la herramienta RADS.

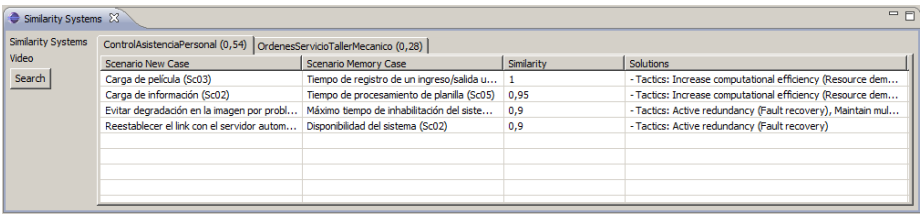


**Fig. 3.** Vista resultante de calcular la similitud entre sistemas.

Considerando la situación en la que el arquitecto decide seleccionar como soluciones las presentadas por el caso *casoMemAP*, las soluciones a aplicar en *casoNuevo* se presentan en la Tabla 6. La aplicación de estas soluciones implica la creación de manera automática de decisiones de diseño (instancias de *ArchitecturalDesignDecision* en Figura 2) asociando para escenario concreto (segunda columna de Tabla 6) las soluciones obtenidas (tercer columna de Tabla 6).

**Tabla 6**. Soluciones aplicadas como resultado de la técnica de RBC.

| Esc. | Breve Descripción | Soluciones Arquitectónicas a Aplicar |
|---|---|---|
| **E01** | Evitar ingreso de usuarios no autorizados bloqueando el acceso a la aplicación. | No se han propuesto soluciones |
| **E02** | Cargar la información contextual de la película en no más de 3 segundos | Tácticas arquitectónicas:<br>- Incrementar recursos disponibles<br>- Incrementar eficiencia computacional<br>- Reducir overhead computacional |
| **E03** | Cargar la película en no más de 3 segundos. | Tácticas arquitectónicas:<br>- Incrementar eficiencia computacional<br>- Reducir overhead computacional |
| **E04** | Comenzar a reproducir la película una vez recibido el requerimiento en no más de 1 segundo. | No se han propuesto soluciones |
| **E05** | Evitar la degradación de la imagen por problemas de carga de la red. | Tácticas arquitectónicas:<br>- Activar redundancia |
| **E06** | Restablecer el link con el servidor en no más de 5 segundos en casos de que ocurra alguna falla. | Tácticas arquitectónicas:<br>- Activar redundancia<br>- Mantener múltiples copias de los datos<br>- Monitorear procesos<br>- Utilizar transacciones |
| **E07** | No requerir de entrenamiento para que cualquier usuario pueda utilizar la aplicación. | No se han propuesto soluciones |

## 4 Trabajos relacionados

Brindar soporte al diseño arquitectónico de sistemas de software es una tarea difícil, debido a que depende fuertemente de aspectos humanos, como son la experiencia y el conocimiento de los arquitectos involucrados. Existen numerosas contribuciones que trabajan sobre la captura del razonamiento realizado por los arquitectos al tomar las decisiones de diseño de manera que sea de utilidad en etapas posteriores del desarrollo del sistema e inclusive en el diseño de nuevas arquitecturas. Algunas de éstas son detalladas en [12], [13], [14], [15], [16]. Sin embargo, no describen claramente cómo puede ser reutilizada una solución arquitectónica en el diseño de nuevas arquitecturas.

Bachmann y otros [17] han desarrollado un sistema basado en reglas, llamado ArchE (Architecture Expert), el cuál sirve como un asistente para el arquitecto de software. ArchE trabaja con "frameworks" de razonamiento de atributos de calidad para predecir respuestas de la arquitectura en una situación dada. Requiere de uno o más "frameworks" de razonamiento para cada atributo de calidad, aunque muchos de ellos aún no han sido creados. Sin embargo, la contribución realizada en este trabajo podría ser integrada con esta herramienta para asistir al diseñador a generar una arquitectura inicial en base a experiencias pasadas, que luego sería refinada y detallada utilizando las sugerencias realizadas por ArchE.

Monroe y Garlan [1] describen una manera de diseñar arquitecturas de software reutilizando diseños arquitectónicos, en dónde se utilizan los estilos arquitectónicos para soportar la clasificación, el almacenamiento y la recuperación de elementos de diseño arquitectónicos reusables. Proponen una herramienta que asista a los diseñadores seleccionando los elementos de diseño y los patrones apropiados basándose en información estilística, a diferencia de la presente propuesta cuyo objetivo es asistir a los arquitectos brindándole un conjunto de soluciones arquitectónicas (estilos y tácticas) utilizadas previamente para satisfacer un conjunto de requerimientos de calidad ingresado.

Por otro lado, la idea de utilizar técnicas de RBC para asistir o automatizar el proceso de diseño es tan antigua como el campo de RBC mismo [18]. La técnica de RBC ha sido utilizada como soporte al diseño en otras disciplinas, como ser: diseño arquitectónico de edificios, diseño de configuración de automóviles, diseño de planes de comida, etc.

En el área de diseño arquitectónico, Vazquez y otros [19] han utilizado el RBC para derivar implementaciones orientadas a objetos a partir de arquitecturas de software proponiendo una herramienta que permite asistir al diseñador. En esta propuesta se trata de asistir al diseñador para confeccionar un diseño detallado del sistema partiendo del diseño arquitectónico, lo cuál difiere del trabajo presentado.

Por otro lado, Gomes y Leitão han aplicado una técnica de RBC para reutilizar el conocimiento en el diseño de software ([20], [21]). Su propuesta, en la última versión, consiste en reutilizar diagramas de clases de UML, o partes de ellos, que describan el diseño de un sistema. La similitud entre los diagramas es calculada considerando las similitudes de sus elementos, los cuales pueden ser paquetes, clases o interfaces. Dichas similitudes están expresadas en similitudes de conceptos (para lo cuál aplican ontologías), y de relaciones con otros elementos.

## 5 Conclusiones

En esta contribución se presenta la aplicación de una técnica de razonamiento basado en casos para brindar soporte a la reutilización de soluciones arquitectónicas. Los casos son definidos describiendo, por un lado, el conjunto de requerimientos de los "stakeholders" del sistema y catalogando los requerimientos de calidad a través de la utilización de escenarios de atributos de calidad (generales y concretos), y por el otro las soluciones arquitectónicas aplicadas por los arquitectos para satisfacer dichos requerimientos. Estas soluciones arquitectónicas pueden ser estilos arquitectónicos, tácticas arquitectónicas o soluciones aplicadas por los arquitectos a partir de experiencias propias.

La similitud entre dos casos (sistemas) es medida comparando los escenarios concretos de atributos de calidad de cada uno de ellos, teniendo en cuenta los escenarios generales de los cuales son instancias, sus prioridades y el grado de satisfacción del escenario en función de las soluciones aplicadas.

También se describe una herramienta desarrollada, llamada RADS, que permite utilizar la técnica previamente definida.

Esta contribución trabaja en un área que se encuentra aún en crecimiento y puede ser integrada con otras propuestas para brindar una solución integral al diseño de un sistema.

Los trabajos futuros refinarán el modelo propuesto con el objetivo de considerar, al evaluar la similitud de los casos, las restricciones que podrían surgir como necesidades específicas de los "stakeholders", como ser el requerimiento de la utilización de determinada tecnología, o determinado componente, o distribución física.

**Agradecimientos.** Este trabajo ha sido financiado en forma conjunta por CONICET, la Universidad Tecnológica Nacional y la Agencia Nacional de Promoción Científica y Tecnológica (PAE-PICT 2315, IP-PRH 2007). Se agradece el apoyo brindado por estas instituciones.

## Referencias

1. Monroe, R., Garlan, D.: Style-Based Reuse for Software Architectures. Proceedings of the 4th International Conference on Software Reuse, pp. 84 (1996).
2. Garlan, D., Shaw, M.: An introduction to software architecture. Technical Report: CS-94-166 (1994).
3. Bass, L., Clements, P., Kazman, R.: Software Architecture In Practice, 2 edition. Addison Wesley (2003).
4. Bachmann, F., Bass, L., Klein, M.: Illuminating the Fundamental Contributors to Software Architecture Quality. Technical Report CMU/SEI-2002-TR-025 (2002).
5. Maher, M. L., Gómez de Silva Garza, A.: Case-Based Reasoning in Design. IEEE Expert, pp. 34-41 (1997).

6. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications. IOS Press. Vol. 7:1, pp: 39-59 (1994).
7. López de Mántaras, R., Plaza, E.: Case-Based Reasoning: an overview. AI Communications. IOS Press. Vol. 10: 1/1997, pp. 21-29 (1997).
8. Watson, I., Marir, F.: Case-Based Reasoning: A Review. The Knowledge Engineering Review. Vol. 9:4 (1994).
9. Bass, L., Klein, M., Bachmann, F.: Quality Attribute Design Primitives. Technical Note CMU/SEI-2000-TN-017 (2000).
10. IBM Rational Software Architect, http://www-01.ibm.com/software/awdtools/swarchitect/index.html (2008)
11. IBM ILOG CPLEX Optimizer, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ (2009)
12. Capilla, R., Nava, F., Pérez, S., Dueñas, J.C.: A web-based tool for managing architectural design decisions. In: Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge (2006).
13. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: Proceedings Fifth IEEE/IFIP Working Conference on Software Architecture, pp. 109-120 (2005).
15. Ali Babar, M., Gorton, I., Kitchenham, B.: A framework for supporting architecture knowledge and rationale management. In: Dutoit, A. H., McCall, R., Mistrik, I., Paech, B. (Eds.), Rationale Management in Software Engineering. Springer, pp. 237-254 (2006).
14. Tang, A., Jin, Y., Han, J.: A rationale –based architecture model for design traceability and reasoning. Journal of Systems and Software 80, pp. 918-934 (2007).
17. Bachmann, F., Bass, L., Klein, M.: Preliminary Design of ArchE: A Software Architecture Design Assitant. Technical Report CMU/SEI-2003-RT-021 (2003).
16. Carignano, M. C., Gonnet, S., Leone, H.: A Model to Represent Architectural Design Rationale. In: Working IEEE/IFIP Conference on Software Architecture 2009 (WICSA 2009 ) & European Conference on Software Architecture 200, pp: 301-304 (2009).
18. Maher, M. L.,Pearl, P.: Issues and Applications of Case-Based Reasoning in Design. Lawrence Erlbaum Associates, Inc. (1997)
19. Vazquez, G., Diaz Pace, J. A., Campo, M.: Reusing design experiences to materialize software architectures into object-oriented designs. Information Sciences. Elsevier (2010).
20. Gomes, P., Pereira, F., Paiva, P., Seco, N., Carreiro, P., Ferreira, J., Bento, C.: REBUILDER: A CBR Approach to Knowledge Management in Software Design. In: Advances in Learning Software Organizations, pp. 31-42 (2004).
21. Gomes, P., Leitão, A.: A Tool for Management and Reuse of Software Design Knowledge. S. Staab and V. Svatek (Eds.): EKAW 2006, LNAI 4248, pp. 381-388 (2006).