Universidad Tecnológica Nacional
Septiembre-Diciembre 2018 / Año 16 - Nº 33

Revista Tecnología y Ciencia
DOI: https://doi.org/10.33414/rtyc.33.208-229.2018 ISSN 1666-6933

Un Enfoque Inteligente para Soporte a la Toma de Decisiones de Diseño Arquitectónicas en el Contexto de la Evaluación de Arquitecturas de Software

Verónica Bogado
CIT Villa María (CONICET-UNVM) / Departamento Ingeniería en Sistemas de Información FRVM, UTN -Villa María, Córdoba, Argentina
vbogado@frvm.utn.edu.ar

Eva Villarreal Guzmán
Departamento Ingeniería en Sistemas de Información FRVM, UTN -Villa María, Córdoba, Argentina
villarrealguzman@gmail.com

Silvio Gonnet
INGAR (CONICET – UTN) Santa Fe, Argentina

Horacio Leone
INGAR (CONICET – UTN) Santa Fe, Argentina
hleone@santafe-conicet.gob.ar

Presentación 18/11/2016
Aprobación 20/07/2017

Resumen

La Ingeniería de Software necesita herramientas novedosas para alcanzar alta calidad en el software, enfrentando el rol cambiante del mismo. La Arquitectura de Software es clave, ya que afecta directamente a la calidad final. La Evaluación de Arquitecturas de Software valida si la arquitectura cumple con los requerimientos de calidad, implicando decisiones de diseño. La toma de decisiones es un proceso complejo conducido por factores humanos, donde la Inteligencia Artificial puede asistir. Entonces, se propone un enfoque basado en Inteligencia Artificial para ayudar a arquitectos en el proceso de toma de decisiones de diseño conducido por atributos de calidad. Esta versión combina modelos de atributos de calidad y un Agente inteligente, utilizando Aprendizaje por Refuerzo para obtener una política de aplicación de patrones arquitectónicos secuencial mediante simulación. Un estudio de caso y una serie de experimentos ilustran la propuesta con patrones comúnmente utilizados en la industria del software.

Palabras clave: Evaluación de Arquitecturas de Software, Toma de Decisiones de Diseño, Inteligencia Artificial.

Abstract

Software Engineering needs novel tools to pursue further the goals of achieving software quality, facing the changing role of software. In this context, Software Architecture plays a key role because it directly affects the final quality. Software Architecture Evaluation validates if the architecture achieves the quality requirements, and triggers a set of design decisions. The decision-making is a very complex process driven by several human factors. Nowadays, it is argued that Artificial Intelligence-based practices can assist this process. In this work, an Artificial Intelligence-based approach for assisting architects in the design decision-making process driven by quality attributes is proposed. This first version combines quality-attribute models and an intelligent agent to support software architecture evaluation. It applies Reinforcement Learning tools to obtain a sequential architectural pattern application policy by simulation. A case study and a set of experiments illustrate the proposal with patterns commonly used in software industry.

Keywords: Software Architecture Evaluation, Design Decision-making, Artificial Intelligence.

Introducción

El Software es un sistema transversal que impacta en las actividades de cualquier organización incluyendo empresas, gobiernos y sociedades en general. En los últimos años, la demanda por parte de usuarios y desarrolladores de software de mayor funcionalidad, seguridad, performance, usabilidad, y muchos atributos más, impulsa la generación de innovación y nuevos paradigmas. Estos desafíos requieren una respuesta rápida en términos de metodologías y herramientas para concebir un software de alta calidad.

En este contexto, el papel de la calidad de software se convierte en un tema fundamental para su ciclo de vida (Nielsen, 2015), donde el diseño de la Arquitectura de Software se vuelve una práctica central durante el desarrollo, debido a la creciente complejidad de los sistemas y su impacto en la calidad. La Arquitectura de Software (AS) es el medio por el cual se alcanzan los atributos de calidad y se gestionan los riesgos y costos en proyectos tecnológicos complejos. La AS limita el nivel de los atributos de calidad, siendo un puente entre los requerimientos y el diseño detallado o la implementación (Bass, Clements and Kazman, 2013) e incluso es un Plan de Diseño para el desarrollo del producto (Hofmeister, Nord, and Soni, 2000).

El diseño de arquitecturas implica análisis, síntesis y evaluación. Los arquitectos deben tomar decisiones respecto a patrones arquitectónicos, tácticas y descomposición de funcionalidad a alto nivel con el fin de razonar sobre las soluciones potenciales. Las prácticas basadas en la Inteligencia Artificial (IA) pueden ayudar a los desarrolladores a buscar en el espacio de diseño más efectivamente. Las mismas

Revista Tecnología y Ciencia
ISSN 1666-6933

también han mostrado ser útiles a la hora de apoyar el proceso de toma de decisiones de diseño (Meziane, and Vadera, 2010).

En este trabajo, se propone un enfoque basado en IA para asistir a los arquitectos en el proceso de toma de decisiones de diseño conducido por atributos de calidad. Esta primera versión del enfoque combina modelos de atributos de calidad y un Agente Inteligente para dar soporte a la evaluación de la AS. A diferencia de otras propuestas, este enfoque se basa en un modelo de calidad (conducido por métricas) y aplica herramientas de Aprendizaje por Refuerzo (AR, o RL en inglés) para obtener una política de aplicación de patrones arquitectónicos secuencial mediante simulación. Esta política permite analizar en tiempo real el impacto de la aplicación de una secuencia de patrones sobre la calidad del software cuando una arquitectura está siendo evaluada. La propuesta se centra en un enfoque de calidad cuantitativo basado en métricas que son indicadores de atributos de calidad visibles en tiempo de ejecución. Esta perspectiva permite integrar enfoques de evaluación de arquitecturas de software basado en métricas con un soporte al diseño basado en indicadores de calidad. Por lo tanto, el aprendizaje del Agente se logra mediante la experiencia de la aplicación de un patrón y el impacto en la calidad de un sistema particular, utilizando este conocimiento en nuevos proyectos.

Trabajos relacionados

En el diseño arquitectónico, el análisis, síntesis y evaluación se realizan incremental e iterativamente. La industria de software necesita mejorar la relación entre la documentación de la AS y su validación. Estudios han mostrado cómo los requerimientos no funcionales afectan las decisiones arquitectónicas y la relevancia de la Performance, Confiabilidad y Usabilidad en el producto de software. Además, se ha enfatizado la importancia de expresar los requerimientos no funcionales de forma medible para validarlos tempranamente y evitar pruebas que consuman tiempo o sean imposibles de ejecutar (Ameller et al, 2013).

La evaluación de la AS implica el análisis de diferentes atributos de calidad (fracción de los requerimientos no funcionales), donde los externos son visibles en tiempo de ejecución y los internos se focalizan en aspectos estáticos. Muchos enfoques proponen la evaluación de atributos de calidad basada en un análisis cualitativo o cuantitativo (Clements, Kazman, and Klein, 2002; Wen-Li Wang, Dai Pan, and Mei-Hwa Chen, 2006; Sharma and Trivedi, 2007; Fukuzawa and Saeki, 2012; Rathfelder et al, 2013; Christensen and Hansen, 2010). Además, existen trabajos focalizados en mejorar la especificación de la AS para su evaluación (Brosch, 2012). Otros enfoques intentan proporcionar un análisis trade-off entre varios atributos de calidad visibles en tiempo de ejecución, donde el software se analiza incluyendo calidad y funcionalidad (Bogado, Gonnet y Leone, 2014). En todos los casos, luego de la evaluación de la AS, si la misma no fuese validada, se deberían tomar decisiones de diseño, las cuales implican aplicación de patrones arquitectónicos, estrategias de implementación y plataformas tecnológicas (Ameller et al, 2013).

La toma de decisiones arquitectónicas se vuelve complicada cuando involucra muchos requerimientos desde múltiples perspectivas y existen muchas soluciones que satisfacen los mismos requerimientos, y cada una de estas soluciones implica

un análisis trade-off entre varios atributos de calidad (Meiziane and Vadera, 2010). Además, los arquitectos no siguen una técnica de toma de decisiones sistemática. En cambio, sus decisiones se basan principalmente en características personales como la intuición y la experiencia, siguiendo enfoques informales pero estructurados para la toma de decisiones. Finalmente, hay muchos factores incluyendo el tiempo, dinero y prácticas organizacionales que hacen difícil a los responsables de las decisiones tomarlas, debiendo realizar análisis extensos antes (Dasanayake et al, 2015).

En los últimos años, las técnicas de IA han sido utilizadas para brindar soporte a tareas del proceso de desarrollo de software. En particular, las redes Bayesianas y la Lógica Difusa fueron utilizadas para analizar requerimientos; Razonamiento Basado en Casos (RBC) y Programación por Restricciones han sido utilizadas para diseñar y Algoritmos Genéticos (AG) han sido aplicados para codificar y testear. Específicamente, en el diseño de la AS, las técnicas de IA emplean atributos de calidad para definir una función de bondad sobre el espacio de arquitecturas posibles; luego, se busca en el espacio de posibles descomposiciones jerárquicas de un sistema (Ammar, Abdelmoez y Hamdi, 2012).

Existe claramente una tendencia creciente de la Ingeniería Inteligente. El Aprendizaje Automático puede incrementar el poder de los sistemas de software y ayudar a los desarrolladores a aprender sobre problemas y dominios. El Aprendizaje por Refuerzo introduce la exploración y explotación para exhibir el comportamiento de aprendizaje inteligente que se espera en muchos problemas complejos y de la vida real (Kulkami, 2012). Algunos autores sugieren que el orden en que las decisiones se toman debe tener un impacto en el resultado (el problema que se está resolviendo) y la primera decisión debe ser particularmente importante (Van Vliet and Tang, 2016). De esta forma, las prácticas del AR permiten a los desarrolladores modelar y simular este tipo de problemas (Kulkarni, 2012).

Soporte inteligente a las decisiones arquitectónicas

El diseño de la AS es un proceso clave para el desarrollo de software y para el éxito de los proyectos en las empresas de software. Este proceso incluye un conjunto de actividades complejas debido a la naturaleza del diseño de software y la complejidad de decidir por una alternativa sobre otras soluciones, aún más considerando que la calidad de la solución debe alcanzar los requerimientos del usuario.

Las actividades generales propuestas en (Hofmeister et al, 2007) son Análisis, Síntesis y Evaluación (Figura 1). Este trabajo se centra en las últimas dos actividades. La síntesis arquitectónica propone soluciones arquitectónicas para una serie de requerimientos; por lo tanto, se mueve desde el problema hacia el espacio de solución. La evaluación arquitectónica asegura que las decisiones sobre el diseño arquitectónico sean las correctas, es decir, que asegura que la AS alcance los requerimientos de calidad

Existe una relación directa entre la síntesis y la evaluación. En consecuencia, es importante prestar atención explícita a las decisiones en la evaluación de la arquitectura (Van Vliet y Tang, 2016). Si la AS no alcanzase los atributos de calidad en la medida requerida, se deberían tomar nuevas decisiones arquitectónicas.

Revista Tecnología y Ciencia
ISSN 1666-6933

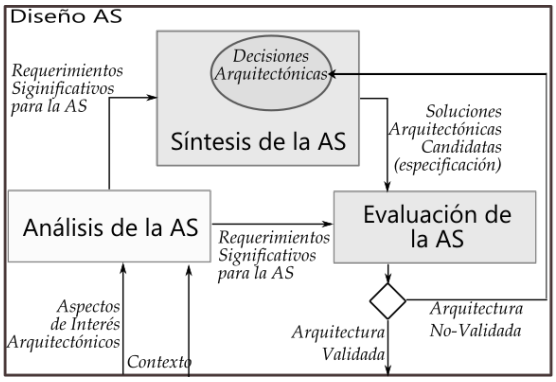


Figura 1. Un modelo general de diseño de arquitecturas de software, adaptado de (Hofmeister et al, 2007).

Aunque existen herramientas para especificar AS, estas herramientas no brindan soporte al arquitecto en la toma de decisiones informadas conducidas por medidas de los atributos de calidad. Este trabajo propone un Agente (Asistente de diseño) para recomendar el uso de algunos patrones para satisfacer escenarios de atributos de calidad, pero considerando también escenarios funcionales, es decir, la asignación de responsabilidades a los componentes.

Análisis y especificaciones de la arquitectura

Las primeras entradas son los Requerimientos Funcionales (RFs) y los Requerimientos No Funcionales (RNFs); particularmente, los Requerimientos de Calidad (RCs) relacionados con los Atributos de Calidad (ACs). Los RFs describen los escenarios del sistema, los cuales definen las responsabilidades de cada componente de software. Los RCs especifican las respuestas del software para concretar los objetivos de negocio. Es importante especificar claramente el valor de cada requerimiento de calidad para validar cuantitativamente. Una técnica interesante son los Escenarios de Atributos de Calidad (en inglés, QAS) (Bass, Clements and Kazman, 2013).

Muchos factores son sintetizados para obtener la especificación de la AS utilizando la notación Use Case Map (UCM). Este modelo representa la AS y escenarios de sistema. Esta notación visual permite a los arquitectos modelar sistemas dinámicos complejos haciendo más fácil el trabajo de analizar la AS. La razón principal para la elección de esta notación es que UCM construye escenarios funcionales junto con estructuras de software mediante el solapamiento de caminos compuestos por responsabilidades y otros elementos, todo en el mismo modelo. Esta notación representa unidades presentes en tiempo de ejecución (visión dinámica del software). Esta visión permite a los arquitectos analizar atributos de calidad que son visibles durante la operación del software. Los elementos básicos de esta notación son (ITU-T Z 151, 2012):

- Responsabilidad: es una acción de alto nivel de un componente, el cual es responsable de ejecutar.

- Equipo (Team): representa a ambos tipos de componentes de software, simples y compuestos. Es una entidad de alto nivel que puede agrupar otras entidades más pequeñas.
- Punto de Partida: comienzo de un escenario (o ruta o camino) por medio de un estímulo.
- Punto de Fin: fin de un escenario (o ruta o camino).
- Ruta (camino): escenario del sistema, conecta puntos de partida, responsabilidades y puntos de fin.

Actualmente, las empresas de software deben construir una base de datos acerca de la información de calidad de los productos desarrollados durante la ejecución de los proyectos para mejorar la calidad del producto y, en consecuencia, el proceso de desarrollo de software. Por otro lado, el uso de patrones es muy beneficioso para mejorar la calidad de las soluciones de diseño y para acelerar el proceso de desarrollo de software. En el presente trabajo, una decisión arquitectónica está dada por la aplicación de un patrón arquitectónico, ya que implica seleccionar una de las soluciones alternativas. El arquitecto toma una decisión asociada con el patrón en un contexto específico utilizando implícitamente el conocimiento de calidad adquirido en la aplicación previa. Por ejemplo, al considerar dos patrones alternativos: Model-View-Controller o N-Tier; cada uno impacta en los atributos modificabilidad, seguridad y performance. La decisión no es sólo conducida por características estructurales del patrón sino también por el conocimiento implícito del impacto del mismo en la calidad final del sistema. Por los tanto, es importante que las empresas puedan capturar este know-how para convertirlo en un activo de la organización. Capturar esta información en un asistente de software proporcionaría un mecanismo de decisión para la evaluación de arquitecturas de software que es independiente de la persona, manteniendo este conocimiento en la organización. Otro aspecto importante es que los patrones arquitectónicos pueden catalogarse, lo cual facilita la automatización del proceso de selección (Bass, Clements y Kazman, 2013).

Evaluación de arquitecturas

La evaluación de la AS necesita información de calidad. Las métricas e indicadores tienen un papel importante para una evaluación objetiva; así, los arquitectos pueden tomar decisiones estratégicas bien fundadas. Existen muchas prácticas para realizar la evaluación, pero debido al enfoque cuantitativo de esta propuesta, se sugiere simulación ((Bogado, Gonnet y Leone, 2014) o similar) para entrenar y consultar al Agente. Dicha simulación proporciona los indicadores de calidad utilizados por el Agente Inteligente para proponer ajustes en la AS a través de una sugerencia de mejora basada en patrones. Es importante considerar el concepto trade-off, donde la mejora de un aspecto de calidad se presenta en perjuicio de degradar a otro, como es el caso de modificabilidad versus performance. En este trabajo, se consideran varios indicadores, los cuales permiten a los arquitectos analizar parcialmente los siguientes atributos de calidad: performance, confiabilidad, disponibilidad y el impacto en la usabilidad.

Revista Tecnología y Ciencia
ISSN 1666-6933

Métricas de calidad para Trade-off: Las características (atributos) consideradas siguen el estándar ISO25000 siendo: Eficiencia (comportamiento temporal), Confiabilidad (disponibilidad), y su influencia sobre la calidad en uso para los usuarios primarios. Estas características y métricas definidas para medirlas fueron elegidas teniendo en cuenta: la disponibilidad de los recursos humanos para recolectar datos, la facilidad con la que se recopilan, el número de usuarios de los productos de información que utilizan el indicador, la repetitividad y reproductibilidad de los elementos de medida de calidad y la entidad objetivo sobre la cual la información es mantenida (ISO/IEC 25000, 2014).

La Ecuación (1) muestra la definición de la métrica Tiempo de Respuesta del Sistema (System Turnaround Time- STT) tomado como indicador de comportamiento temporal y la Ecuación (2) define la métrica Fallas del Sistema (System Failures-SF) tomada como indicador de disponibilidad, con información de confiabilidad.

$$STT = ((rtt_{1,1} + rtt_{1,2} +...+ rtt_{1,n}) +...+ (rtt_{m,1} + rtt_{m,2} +...+ rtt_{m,n}))/m. \quad (1)$$

Donde rtt (responsibility turnaround time) es el tiempo de respuesta de la responsabilidad, n es el número total de responsabilidades involucradas en el escenario del sistema y m es el número total de solicitudes enviadas por los usuarios para ser procesadas.

$$SF = ((rf_{1,1} + rf_{1,2} +...+ rf_{1,n}) +...+ (rf_{m,1} + rf_{m,2} +...+ rf_{m,n}))/m. \quad (2)$$

Donde rf (responsibility failure) es la falla en la responsabilidad, n es el número total de responsabilidades involucradas en el escenario del sistema y m es el número total de solicitudes enviadas por los usuarios. Una responsabilidad puede fallar solo una vez por solicitud, suponiendo que cuando la falla ocurre en una ejecución continúa la ejecución de la siguiente responsabilidad, es decir, rf= 0 (sin falla) o 1 (falla). Una falla podría causar problemas en las responsabilidades sucesoras debido a la relación causal.

Agente inteligente para las decisiones de diseño

El AR (o RL en inglés) está inspirado en la psicología conductista y se enfoca en cómo los agentes deben tomar decisiones en un entorno dinámico maximizando alguna noción de recompensa acumulada. Un Agente basado en AR es capaz de aprender de la interacción con un entorno explorando y explotando (Figura 2) el conocimiento conducido por una meta. El Agente explota lo que sabe para obtener una recompensa, pero también puede explorar para realizar mejores selecciones de acciones en el futuro (Kulkami, 2012). En el problema de toma de decisiones de diseño arquitectónico, el Agente necesita conocer las arquitecturas especificadas con la notación UCM (entorno que representa la dinámica del software), explorar nuevas Decisiones de Diseño (patrones) y exhibir inteligencia en escenarios nuevos o conocidos (UCM con datos de calidad). ε-greedy es un método de exploración simple, donde el Agente elige la acción que cree que tiene el mejor efecto a largo plazo con probabilidad 1 – ε, o de lo contrario elige una acción uniformemente al azar (Sutton y Barto, 1998). ε es un número real, cuyos posibles valores están entre 0 y 1 inclusive.

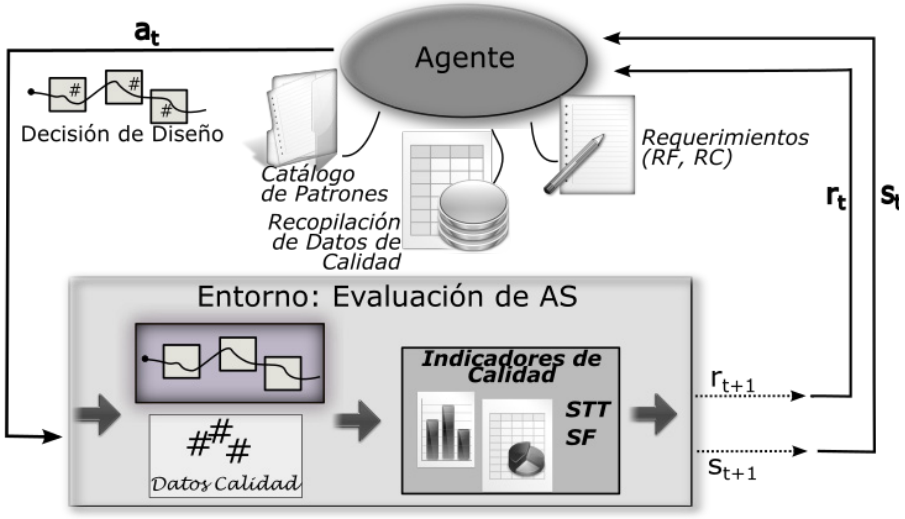


Figura 2. Toma de decisiones de diseño arquitectónico usando el enfoque RL (AR).

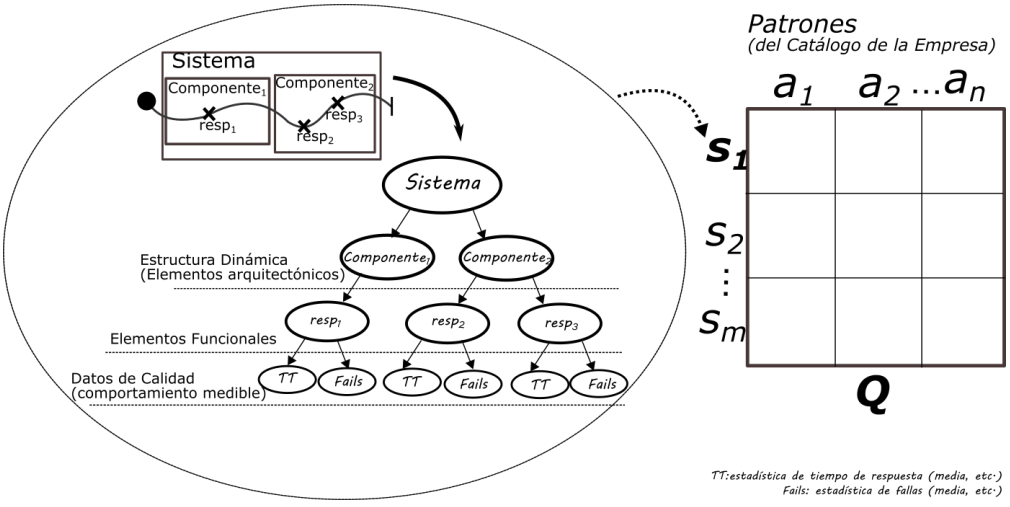


Figura 3. Ejemplo de un estado (s), acción (a) y Q.

El entorno generalmente se formula como un Proceso de Decisión de Markov (PDM, o en inglés MDP) (Sutton y Barto, 1998), (Kulkami, 2012). En este problema de decisión, el entorno representa la dinámica del software a través de su solución de diseño arquitectónico evaluada y ejecutada (Figura 2). Los indicadores de calidad obtenidos durante la evaluación se utilizan como recompensa (r) asociada a la aplicación del patrón seleccionado (acción a) en un tiempo de decisión t. Esta recompensa (r) retroalimenta al Agente; por lo tanto, cuando el mismo percibe un estado similar, el patrón con la mejor recompensa acumulada tendrá más posibilidades de ser seleccionado.

Revista Tecnología y Ciencia
ISSN 1666-6933

Un estado (s) captura la arquitectura del software especificada como UCM en tiempo t (Figura 2). Un ejemplo de un estado se ilustra en la Figura 3 como s1. El modelo UCM está compuesto por elementos de software dinámicos incluyendo componentes simples y compuestos (estructura dinámica), elementos funcionales que representan los escenarios del sistema (o caminos) y datos de calidad medibles (metadatos) relacionados con el patrón en el contexto de aplicación. En este caso, los datos recopilados se relacionan con el tiempo de respuesta y las fallas a bajo nivel (responsabilidad).

El aprendizaje resulta en la selección de una acción o una secuencia de acciones (Kulkami, 2012). Una acción (a) es la aplicación de un patrón en el modelo de AS que lleva a otro UCM, el cual incluye el patrón seleccionado. Este patrón es elegido del catálogo de la empresa conformado por la experiencia de la aplicación del patrón en proyectos anteriores, es decir, la recopilación de datos de calidad (Catálogo de Patrones, Figura 2).

El Agente recibe una retroalimentación del entorno en tiempo t (Figura 2). La recompensa (reward r, Ecuación (3)) fue definida para poder realizar un trade-off de los atributos de calidad mencionados. Por lo tanto, el primer término considera el indicador de performance STT, el requerimiento (TTReq), es decir, el valor deseado, y la prioridad de este indicador sobre el sistema global. El segundo término considera el indicador de confiabilidad SF (disponibilidad), el requerimiento correspondiente (SFReq) y la prioridad definida para éste.

$$reward = (1-STT/TTReq)*1/TTReqPriority + (1-SF/SFReq)*1/SFReqPriority. \quad (3)$$

La prioridad debe ser definida con una escala descendiente. Dos ejemplos comúnmente usados en las empresas de software se definen en la Tabla 1.

Nombre	Valor	Significado
Escala 1	**1-***High (Alta)*	**Requerimiento crítico; obligatorio para el entregable.**
	2-*Medium (Media)*	**Requerimiento eventualmente necesario; se podría esperar hasta una liberación posterior.**
	3-*Low (Baja)*	**Mejora; sería bueno incluirlo, pero si hay suficientes recursos.**
Escala 2	**1-***Essential (Esencial)*	**Debe estar para que el producto sea aceptable.**
	2-*Conditional (Condicional)*	**Mejoraría, pero el producto no es inaceptable si está ausente.**
	3-*Optional (Opcional)*	**Útil, pero el producto es aceptable si está ausente.**

Tabla 1. Escalas de prioridad de requerimientos de calidad.

En resumen, el proceso de Toma de Decisiones del Diseño Arquitectónico, (Figura 4) en el contexto de la evaluación utilizando la notación UCM para especificar las arquitecturas de software, puede considerarse como una secuencia de UCMs. Un diseño monolítico (Monolithic), donde no se aplica ningún patrón, o donde todavía no se ha tomado ninguna decisión, es el primer UCM, el cual debe ser validado contra los requerimientos de calidad (Evaluación). Siguiendo este objetivo de calidad, puede tomarse un conjunto de decisiones de diseño que pase de un UCM a otro, el cual incluye la decisión de diseño aplicada. Este proceso puede modelarse como un PDM (Figura 4), donde hay un estado inicial y en cada estado puede ejecutarse una acción que tiene datos de calidad (aplicación de patrón arquitectónico). Además, una recompensa, la cual incluye a los indicadores de calidad, retroalimenta la decisión pasando a otro estado hasta que el estado objetivo se alcance, es decir, hasta que los requerimientos de calidad se cumplan.

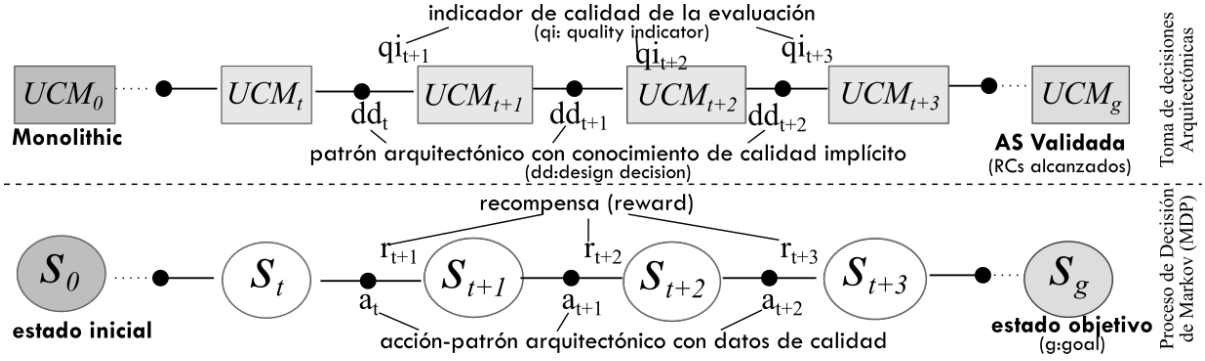


Figura 4. Toma de decisiones arquitectónicas como un modelo MDP.

Los siguientes dos mecanismos de aprendizaje han sido implementados en la propuesta, aplicando una política de exploración-explotación ε-greedy, donde cada episodio consiste en una secuencia de estados alternativa y pares de estado-acción como se muestra en la Figura 4 (Sutton and Barto, 1998).

- Q-Learning: algoritmo de control de diferencias temporales (Temporal Difference-TD) off-policy y, en su forma más simple, se define por regla de actualización:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)].$$

La función acción-valor aprendida (matriz Q en Figura 3) aproxima directamente q*, la función acción-valor óptima, la cual define π* (política óptima) independientemente de la política exploración-explotación seguida (ε-greedy). La política tiene un efecto en el sentido de que determina qué pares de estado-acción son visitados y actualizados. Sin embargo, se requiere para una convergencia que dichos pares sigan siendo actualizados.

La Figura 5 muestra el algoritmo Q-learning, donde el aprendizaje ocurre a través de cada episodio (durante un período) observando un estado s, realizando una

Revista Tecnología y Ciencia
ISSN 1666-6933

acción a y obteniendo una recompensa r del entorno como consecuencia de su actuación en el mismo. En este caso, el Agente aprende la política óptima, es decir, la secuencia de aplicación de patrones óptima independientemente del patrón que utilizan comúnmente la empresa, los arquitectos, o en este caso, el Agente.

```
Initialize Q(s, a) arbitrarily, Q(terminal-state, ·) = 0
Repeat (For each episode):
  Initialize s
  Repeat(For each step of episode)
    Choose a from s using policy derived from Q
    Take action a
    Observe reward r, s'
    Q(s, a)←Q(s, a) + α[r + γmax_A Q(s',A) − Q(s, a)]
    s←s'
  until s is terminal
  (A: set of possible actions in state s', policy: e.g. ε-greedy)
```

Figura 5. Q-learning: algoritmo de control TD Off-policy.

- •SARSA: algoritmo de control de diferencias temporales on-policy, cuya regla de actualización es:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\,[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)].$$

La Figura 6 muestra el algoritmo SARSA. Este mecanismo utiliza experiencias estado-acción-recompensa-estado-acción para actualizar los valores Q (matriz Q en la Figura 3). Esta actualización se realiza luego de cada transición desde un estado no terminal st. Si st+1 es terminal, entonces Q(st+1, at+1) es cero. Como en todos los métodos on-policy, qπ se calcula continuamente para la política de comportamiento π, y al mismo tiempo, cambia la política π seleccionando los mejores valores respecto a qπ. Con este mecanismo de aprendizaje, el Agente aprende una política que consiste en la secuencia de aplicación de patrones elegida con mayor frecuencia durante el proceso de diseño de arquitecturas, es decir, el conjunto de decisiones de diseño que más se repiten, las cuales han sido tomadas luego de la evaluación de arquitectura.

```
Initialize Q(s, a) arbitrarily, Q(terminal-state, ·) = 0
Repeat(For each episode)
  Initialize s
  Choose a from s using policy derived from Q
  Repeat(For each step of episode)
      Take action a
      Observe reward r, s'
      Choose a' from s' using policy derived from Q
      Q(s, a) ← Q(s, a) + α [r + γQ(s', a') − Q(s, a)]
      s←s'
      a←a'
  until s is terminal
                              (policy: e.g. ε-greedy)
```

Figura 6. SARSA: algoritmo de control TD On-policy.

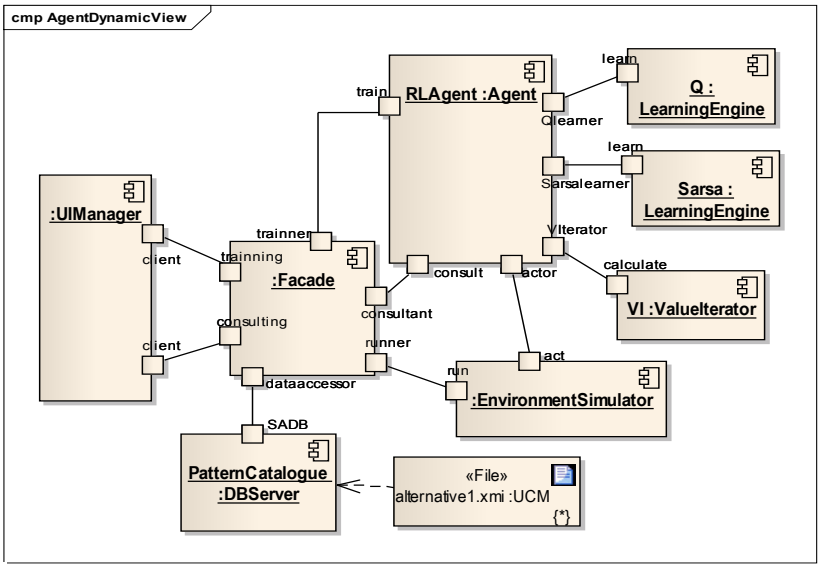


Figure 7. Arquitectura: vista de componentes del prototipo del Agente (RL-Agent).

Implementación

El diseño arquitectónico requiere herramientas computacionales que puedan asistir al proceso de toma de decisiones. Las nuevas herramientas de diseño controlan las condiciones en las que se desarrollan los modelos orientados a objetos, pero muy pocas prestan atención al diseño de arquitecturas del software e incluso a pesar de la importancia a este producto inmediato sobre la calidad del resultado final.

En este trabajo, se diseñó e implementó el entorno de simulación y el Agente utilizando el lenguaje de programación de JAVA. Este prototipo de herramienta para la toma de decisiones de diseño posee dos partes esenciales (Figura 7): Agente y Entorno. El Agente (RLAgent:Agent) presenta dos mecanismos de aprendizaje (Q:LearningEngine y SARSA:LeaningEngine) para resolver el problema arquitectónico. Ambos mecanismos se implementan conforme al motor de aprendizaje del Agente. Un tercer algoritmo, Iteración de Valor (IV, o en inglés VI) (VI:ValueIterator) se implementó para validar el conocimiento generado mediante el uso de los otros mecanismos. El entorno (EnvironmentSimulator) da vida a los modelos UCM. Éste simula el funcionamiento del software siguiendo los parámetros establecidos por el arquitecto en los metadatos del UCM y calcula los indicadores de calidad.

El Agente observa el Entorno y obtiene un estado (UCM) a partir del mismo. El Agente toma una decisión de diseño y el simulador reproduce la ejecución del software mediante el UCM. Luego, el entorno devuelve los indicadores de calidad que han sido tomados de forma dinámica. Tanto el Agente como el Entorno, para representar los estados y elegir el patrón como una acción, requieren de los UCMs. jUCMNav es el editor utilizado para la especificación de los modelos UCM, los cuales incluyen las alternativas de arquitecturas posibles en archivos (File) con formato xmi (jUCMNav, 2016).

Revista Tecnología y Ciencia
ISSN 1666-6933

Esta herramienta funciona bajo el entorno de desarrollo Eclipse. Una Fachada (Facade) estructura a la herramienta y provee una interfaz sencilla para reducir la complejidad del Agente, del Entorno y del Catálogo de Patrones de la empresa. Este patrón simplifica la interfaz del Agente y la legibilidad de la biblioteca de UCMs (PatternCatalogue:DBServer) y reduce dependencias entre diferentes componentes o clientes (UIManager), que proveen una interface gráfica de usuario para mostrar los resultados, lo cual permite al arquitecto analizar el aprendizaje y la aplicación correcta del conocimiento generado por el Agente.

La empresa de software debe retroalimentar la base de datos de UCMs (PatternCatalogue) con los modelos de AS completos, incluyendo la información de calidad necesaria para la simulación del sistema y para calcular los indicadores.

Resultados y discusión

Un estudio de caso y un conjunto de experimentos han sido desarrollados con el propósito de validar la propuesta. El sistema es un software de gestión comúnmente utilizado por la industria de software u otras organizaciones para controlar sus licencias sobre sus sistemas. Este software de gestión de licencias (LM), provisto por la empresa de software, controla dónde y cómo sus productos de software pueden funcionar (se pueden hallar más detalles en (Bogado, Gonnet and Leone, 2014)).

Arquitectura y los requerimientos de calidad

La descripción inicial del sistema considera la RFs y RNFs para construir una arquitectura monolítica que incluya un escenario. Los requerimientos de calidad (Figura 2, dos ejemplos) definen las medidas de respuestas que son utilizadas para evaluar la arquitectura y para tomar decisiones de diseño (Agente).

jUCMNav (2016) se utiliza para la especificación de los UCMs. Esta herramienta permite a los arquitectos especificar los componentes de la arquitectura, la funcionalidad por medio de caminos y los datos de calidad mediante la definición de metadatos (Figura 8). Estos últimos se obtienen de la recopilación de datos (información histórica de proyectos previos). Las especificaciones de los UCMs se guardan como archivos xmi.

ID	Estímulo	Fuente Estímulo	Artefacto	Ambiente	Respuesta	Medida Respuesta	Prior.
RC01	Solicitud	Usuario	*LMSytem*	Normal	Licencia autenticada	Tiempo respuesta<10 seg	*Medium*
RC02	Falla al responder a una solicitud	Responsabilidad	*LMSytem*	Normal	Falla registrada	Total fallas < 1	*High*

Tabla 2. Requerimientos de calidad especificados aplicando Quality Attribute Scenarios (Bass, Clements and Kazman, 2013).

Patrones y datos de calidad

Un conjunto de patrones forma parte de las convenciones de trabajo en cada proyecto de la empresa. Además, en el proceso de diseño, los criterios para la elección de un patrón están basados en un conocimiento implícito sobre los impactos en la calidad del sistema. Las empresas deben ser capaces de distinguir y separar la información de

calidad correspondiente a cada aplicación de un patrón. Esto es una buena práctica que permite a las empresas ser independientes del arquitecto (persona) y compartir este conocimiento entre los miembros del equipo y entre diferentes proyectos.

Los patrones arquitectónicos permiten a los arquitectos gestionar y mejorar la AS, reduciendo la complejidad de la tarea de decidir. En este caso, los patrones utilizados para evaluar el aprendizaje del Agente (conformando el Catálogo para el proyecto) son (Fowler, 2016; Buschmann et al, 1996; Buschmann et al, 2007):

- Broker: es responsable de coordinar la comunicación, reenviando solicitudes o transmitiendo resultados.
- Monolítico (Monolithic): no existen componentes separados arquitectónicamente y la funcionalidad está toda entrelazada. No promueve ningún atributo de calidad.
- Pipe-Filter: transforma los datos de entrada. El Filtro (Filter) modifica la información. Los datos circulan desde un Filtro a otro a través de las Tuberías (Pipe).
- Cliente- Servidor (Client-Server): el sistema es organizado como un grupo de servicios que son proporcionados por componentes llamados servidores, y otros llamados clientes que acceden a esos servicios

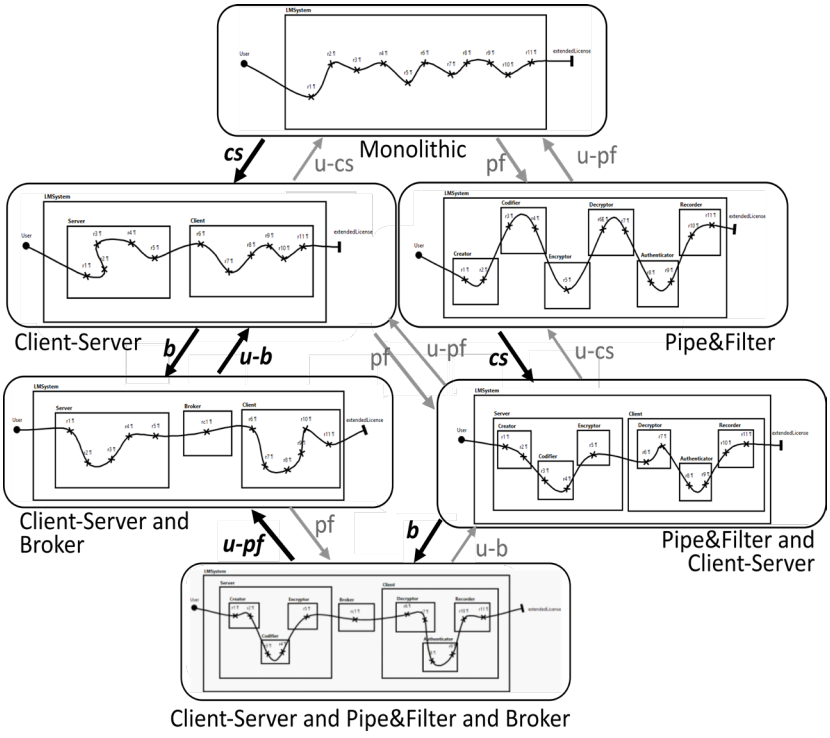


Figura 8. Diagrama de transición de estados con las acciones posibles para el sistema LMSystem.

Revista Tecnología y Ciencia
ISSN 1666-6933

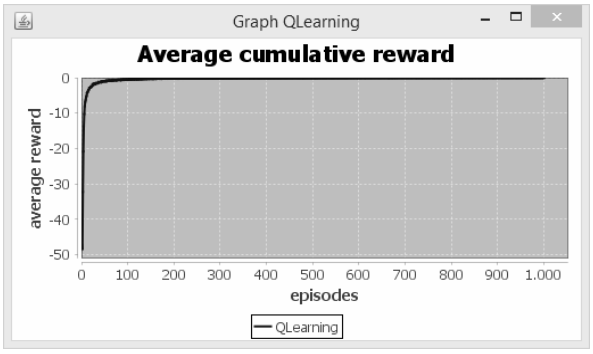
Evaluación y decisiones de diseño

Los UCMs son mapeados a modelos de Markov mediante una transformación automática (directamente de modelo a modelo). Luego, se elige el algoritmo entre Q-Learning y SARSA. Los parámetros de los dos algoritmos están establecidos en la siguiente formula: ε= 0,2, γ= 0,2, α=0,6, y episodes=1000.

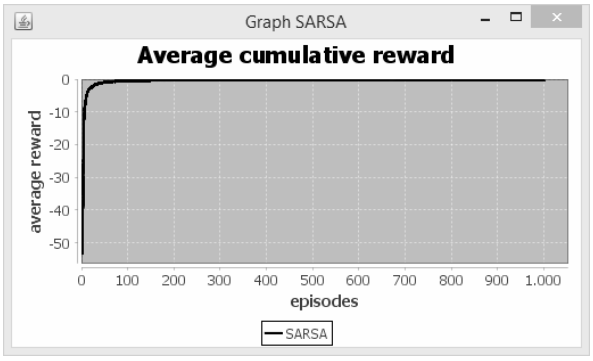
La evaluación es iniciada simulando la ejecución del software, dando vida al modelo UCM (vista dinámica del sistema). El comportamiento de la responsabilidad se simula siguiendo la información de entrada (medias, distribuciones de probabilidad, etc.). Los indicadores de calidad son obtenidos y utilizados para calcular la recompensa para cada estado-acción. El Agente prueba aplicar los patrones, cuyas recompensas permiten el cumplimiento de los RCs, o aproximarse lo más posible a los mismos (r>=0).

La política óptima está dada por la mejor acción que podría ser elegida en cada estado. En la Figura 8, se resalta la política hallada aplicando Q-learning (flechas resaltadas en color negro). Las curvas de aprendizaje se muestran en la Figura 9 para el primer escenario. La primera curva de aprendizaje corresponde a la aplicación del mecanismo Q-learning estabilizado antes de los 100 episodios para este ejemplo. El aprendizaje en SARSA fue similar. SARSA comienza con peores recompensas, pero arriba más rápidamente a una solución factible y su recompensa acumulada promedio a lo largo de los episodios se estabiliza más cerca de 0 (RCs alcanzados). Ningún algoritmo pudo alcanzar el RCs, pero ambos encontraron una buena secuencia de decisiones de diseño, la cuales se aproximan a las medidas de calidad esperadas. Esta política es similar para los dos casos. Por último, el algoritmo de Iteración de Valor de la Programación Dinámica (PD, o en inglés DP) (Sutton y Barto, 1998), se implementa para validar los resultados. La última curva presenta el error entre el Q-learning y IV (VI) y cómo es minimizado a lo largo de las iteraciones.

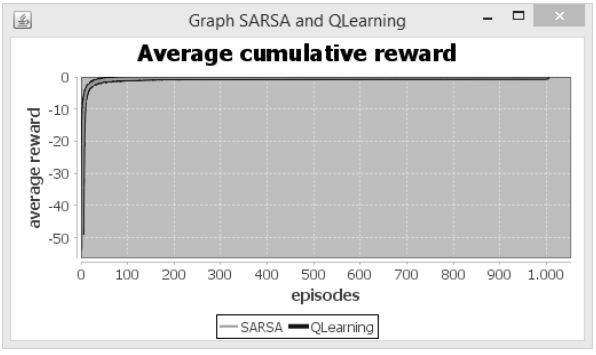
El Agente aprendió sobre las decisiones arquitectónicas mediante experimentos. La política encontrada utilizando ambos algoritmos, Q-learning y SARSA, fue la misma, obteniendo una aplicación de patrones secuencial: Monolitico→Cliente-Server→ Broker como la mejor combinación de patrones para este ejemplo bajo las condiciones dadas por el entorno (Figura 8). El estado que obtuvo el mejor valor fue Cliente-Server y Broker. Este resultado indica que fue el estado en el que los requerimientos de calidad estuvieron más cerca de ser alcanzados. En consecuencia, considerando los resultados en este ejemplo, la política óptima coincide con la solución más elegida.



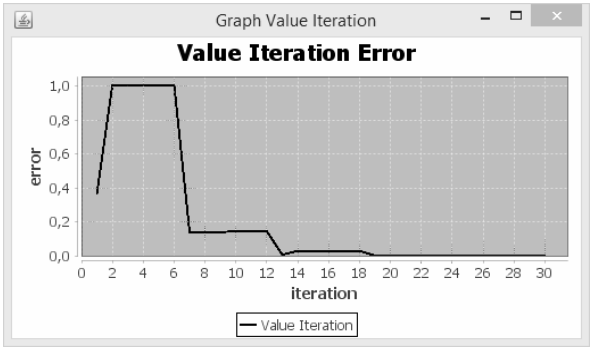
(a) Curva de aprendizaje Q-learning



(b) Curva de aprendizaje SARSA



(c) Curvas de aprendizaje Q/SARSA



(d) V: Curva de error

Figure 9 (a, b, c, d). Curva de aprendizaje obtenidas en los experimentos (Escenario 1).

Revista Tecnología y Ciencia
ISSN 1666-6933

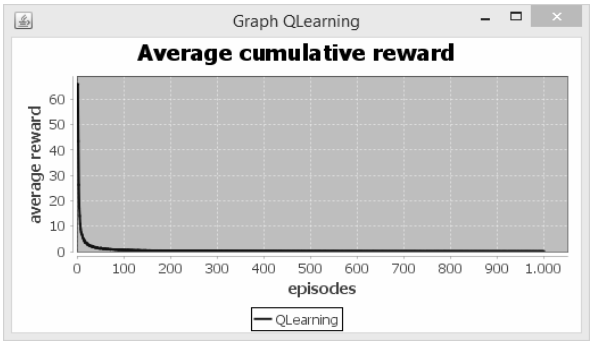
Es importante destacar que, si se considerara una combinación más compleja de patrones e información de calidad, la política podría ser diferente debido al modo en que la recompensa futura se encuentra en la función Q. En Q-learning, es simplemente la acción con el valor más alto la que puede ser elegida a partir de un estado dado, y en SARSA es el valor de la acción real (posiblemente exploratoria) que se tomó en el estado dado. Esto significa que Q-learning permite aprender la política óptima mientras que SARSA tiene en cuenta la política de control realmente empleada por el Agente. En ambos casos, también existe una posibilidad de que se elija una acción aleatoria, dependiendo del mecanismo de exploración incorporado (built-in) en el Agente.

Un segundo escenario fue definido para comprobar el aprendizaje correcto del Agente introduciendo cambios en el entorno durante la simulación. En primer lugar, se modificaron las medias de tiempo del patrón de Pipe-Filter duplicándolas. Esto afectó directamente al indicador STR; por lo tanto, la recompensa fue más negativa. En segundo lugar, en otro experimento, la tasa de fallas del patrón Pipe-Filter se modificaron en una forma similar (afectando el indicador SF). Finalmente, los datos de calidad de Pipe-Filter (medias, tasas de fallas), fueron manipulados para forzar que esta sea la peor acción entre todas las alternativas posibles, obteniéndose los resultados esperados. El Agente aprendió que este estado no es una alternativa bajo estas nuevas condiciones, eligiendo siempre las otras alternativas, dado que este estado tenía un valor Q menor y la peor recompensa (-2,9964).

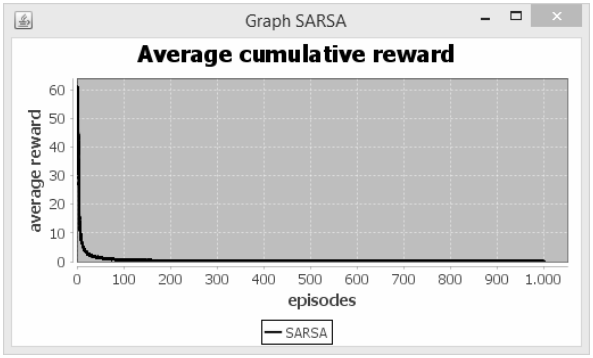
Un tercer escenario se focalizó en los requerimientos de calidad. El experimento introdujo una modificación en las mediciones de respuestas, duplicando los valores para atenuarlos. La Figura 10 muestra las curvas de aprendizaje obtenidas. En este caso, no se alcanzaron los requerimientos, pero se encontró una buena solución, donde las curvas de aprendizaje se estabilizaron muy cerca de 0. La convergencia de las curvas de aprendizaje fue más rápida que en los experimentos anteriores (escenario 1). También, es importante observar las curvas de error, en las Figuras 9 (d) y 10 (d), en donde el error se estabiliza después de la iteración 18 y 13 en el primer y segundo caso respectivamente.

El diseño de la AS es un proceso muy complejo. En particular, la evaluación de la AS implica diversos aspectos como la estructura del software, la calidad (incluidos los requerimientos, las métricas, etc.) y las decisiones de diseño. En este sentido, la toma de decisiones se convierte en una actividad multifacética. Un estado complejo es necesario para representar mejor el dominio, el cual implica: datos estructurales (patrón y otros componentes), de comportamiento (responsabilidad) y de calidad (forma medible). La notación UCM permite a los arquitectos definir todos los aspectos mencionados en el mismo modelo, proporcionando una visión dinámica completa del software en una etapa temprana del desarrollo. Este modelo se ejecuta por simulación, utilizando un conjunto de indicadores de calidad para definir una recompensa que retroalimenta al Agente. Además, los requerimientos de calidad definen el objetivo de aprendizaje y los parámetros de aprendizaje de los algoritmos también pueden alterar los resultados.

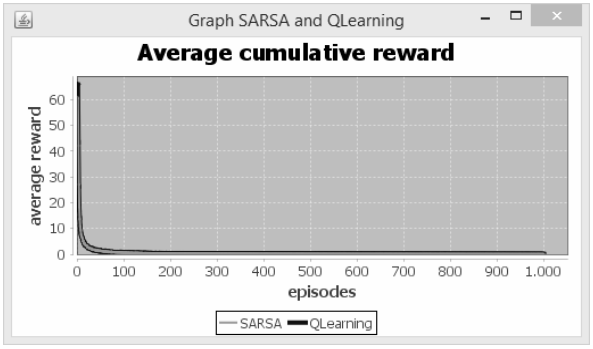
Todas estas variables pueden ser analizadas y, por lo tanto, tienen efectos sobre el aprendizaje y las soluciones propuestas por el Agente. La ventaja es que el enfoque AR le permite al Agente aprender del entorno (de la AS e indicadores de calidad) mientras



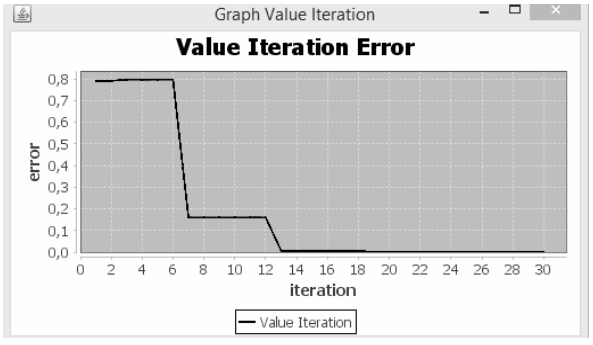
(a) Curva de aprendizaje Q learning



(b)Curva de aprendizaje Sarsa



(c)Curvas de aprendizaje Q/SARSA



(d)IV: Curva de error

Figure 10. Curvas de aprendizaje obtenidas en los experimentos (Escenario 3).

Revista Tecnología y Ciencia
ISSN 1666-6933

Éste continúa variando. El Aprendizaje por Refuerzo difiere del aprendizaje supervisado en el hecho de que los pares entrada-salida correctos no se presentan y la elección de acciones sub-óptimas no es corregida de manera explícita. Además, se presta especial atención al rendimiento on-line, el cual consiste en encontrar un balance entre la exploración (soluciones inexploradas) y explotación (conocimiento adquirido) (Sutton y Barto, 1998). Con el mecanismo Q-learning, el Agente puede adquirir una política óptima independientemente de cómo seleccione las acciones, mientras explore lo suficiente; con SARSA, el Agente puede aprender acerca de las soluciones realmente seleccionadas (basado sólo en su experiencia previa).

En este trabajo, el estado inicial es una arquitectura monolítica, en donde no se aplica ningún patrón arquitectónico. Sin embargo, el Agente puede aprender desde cualquier estado debido a que se trata de lograr un objetivo de calidad, es decir, un análisis trade-off entre los requerimientos de calidad especificados relacionados con ciertos atributos de calidad.

Conclusión y trabajo futuro

En la industria de software, la toma de decisiones es seguramente la tarea más importante para los arquitectos de software y, a menudo, considerablemente difícil. Este trabajo presentó una propuesta basada en técnicas y metodologías de IA para asistir a los arquitectos durante el diseño de arquitecturas de software con el objetivo de dar un apoyo concreto a la etapa de evaluación de la arquitectura. Con esta propuesta, el arquitecto no sólo puede analizar diversos atributos de calidad con el mismo modelo de evaluación, sino que también puede obtener una visión integrada incluyendo aspectos funcionales en la evaluación de la AS. El principal aspecto a destacar es el enfoque de calidad basado en métricas sobre el cual se pueden tomar decisiones de diseño objetivas.

Este trabajo propuso una perspectiva de la AS basada en modelos de calidad. Con esta perspectiva, el Agente puede percibir la estructura del software, el comportamiento y el impacto de la calidad de aplicar un patrón a partir de experiencias anteriores. El Agente aprende de la interacción con el entorno (arquitecturas alternativas e indicadores de calidad obtenidos en la evaluación). Hoy en día, las herramientas de diseño controlan condiciones en modelos detallados, pero no se focalizan ni en la arquitectura del software ni en la evaluación de la misma. Esta propuesta puede ayudar en las actividades de evaluación y mejora de las arquitecturas mediante la simulación de la operación del software, la obtención de indicadores de calidad y las sugerencias de patrones (decisiones de diseño) de acuerdo con los requerimientos de calidad.

Finalmente, se desarrolló un estudio de caso utilizando una AS tradicional y datos históricos a partir de una base de datos de proyectos de una empresa. Se implementaron dos algoritmos AR (Q-learning y SARSA) y se utilizó Iteración de Valor para validar los valores devueltos por los otros dos algoritmos. Es importante destacar la complejidad del proceso de toma de decisiones de diseño arquitectónico, más aún considerando un enfoque basado en métricas y un modelo de calidad de software.

Como trabajo futuro, es interesante capacitar al Agente con sistemas más complejos, incluyendo otros patrones arquitectónicos, nuevos atributos de calidad como seguridad, y/o nuevas métricas para performance, disponibilidad y confiabilidad. Modelos de aprendizaje jerárquicos podrían ser aplicados para mejorar el proceso de toma de decisiones y su combinación con una evaluación de la AS basada en DEVS (Bogado, Gonnet y Leone, 2014). Finalmente, se proyecta el desarrollo de una herramienta que pueda integrarse a un entorno de desarrollo de software utilizado por arquitectos.

Revista Tecnología y Ciencia
ISSN 1666-6933

References

Nielsen, P. D. (2015). "Software Engineering and the Persistent Pursuit of Software Quality", *CrossTalk-Journal of Defense Software Engineering*, May/June.

Bass, L., Clements P. y Kazman, R. (2013). *Software Architecture in Practice.* Addison-Wesley, MA.

Hofmeister, C., Nord, R., Soni, D. (2000). *Applied Software Architecture.* Addison-Wesley Professional.

Meziane, F. y Vadera, S. (2010). *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects*, Information science reference, NY.

Ameller, D., Ayala, C., Cabot, J. y Franch, X. (2013). "Non-functional Requirements in Architectural Decision Making" en *IEEE Software*, March/april: 61-67.

Clements, P., Kazman, R. y Klein, M. (2002). *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, MA.

Wang, W. L., Pan, D. y Chen, M. H. (2006). "Architecture-based Software reliability modeling" en *J. Sys. Software* 79, 1: 132-146.

Sharma, V. S. y Trivedi, K. S. (2007). "Quantifying software performance, reliability and security: an architecture-based approach" en *J. Sys. Software* 80, 4: 493-509.

Fukuzawa, K. y Saeki, M. (2012). "Evaluating Software Architecture by Coloured Petri Nets", en Proceedings 14th International Conference on Software Engineering and Knowledge Engineering, Italy: 263-270.

Rathfelder, C., Klatt, B., Sachs y K.; Kounev, S. (2013). "Modelling Event-based Communication in Component-based Software Architectures for Performance Predictions" en Softw. Sys. Model., 13, 4: 1291-1317.

Brosch, F., Koziolek, H., Buhnova, B. y Reussner, R. (2012). "Architecture-based reliability prediction with the Palladio Component Model" en *IEEE T. Software Eng.*, 38, 6.

Christensen, H. y Hansen, K. (2010). "An empirical investigation of architectural prototyping" en *J. Sys. Software* 83, 1: 133-142.

Bogado, V., Gonnet, S. y Leone, H. (2014). "Modeling and Simulation of Software Architecture in Discrete Event System Specification for Quality Evaluation" en *SIMULATION* 90, 3: 290-319.

Dasanayake, S., Markkula, J., Aaramaa, S. y Oivo, M. (2015). "Software Architecture Decision-Making Practices and Challenges: An Industrial Case Study" en Proc. 24th Australasian Software Engineering Conference, Adelaide, Australia.

Ammar, H., Abdelmoez, W. y Hamdi, M. S. (2012). "Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems", en Proceedings 15th International Conference on Computer and Information

Technology, Bangladesh.

Kulkarni, P. (2012). *Reinforcement and Systemic Machine Learning for Decision Making*. IEEE/Wiley, US.

Van Vliet, H. y Tang, A. (2016). "Decision Making in Software Architecture" en *J. Sys. Software* 7: 1-7.

Hofmeister, C., Kruchten, P., Nord, R., Obbink, H., Ran, A. y America, P. (2007). "A General Model of Software Architecture Design derived from Five Industrial Approaches" en *J Sys Software* 80, 1: 106-126.

ITU-T Z.151 (2012). Series Z, Languages And General Software Aspects for Telecommunication Systems, Formal description techniques (FDT). User Requirements Notation (URN)-Language definition, ITU.

ISO/IEC 25000 (2014). Systems and software engineering--Systems and software Quality Requirements and Evaluation (SQuaRE).

Sutton, R. y Barto, A. (1998). *Reinforcement Learning: an introduction*. MIT Press, MA.

jUCMNav, disponible en <http://jucmnav.softwareengineering.ca>. Consultado el 07-2016.

Fowler, M. (s/a). *Catalog of Patterns of Enterprise Application Architecture.* Disponible en <http://martinfowler.com/eaaCatalog/> consultado el 07-2016.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. y Stal, M. (1996). *Pattern-oriented Software Architecture*, Volumen 1: A System of Patterns. John Wiley & Sons.

Buschmann, F., Henney, K. y Schmidt, D. C. (2007). *Pattern-oriented Software Architecture*, Volumen 5: On Patterns and Pattern Languages. John Wiley & Sons, England.