Using Model-to-Model Transformations for Web Software Architecture Simulation

María J. Blas, and Silvio Gonnet

***Abstract*— The development of evaluation methods that allow improving the capture of quality data regarding software architectures is a topic of interest in Software Engineering. However, the dynamic of the traditional methods used for architecture evaluation is not enough to deal with the architectural complexity exhibit in new types of software products (such as cloud and web applications). In this paper, we present a model-to-model mapping based on modeling and simulation techniques that can be used as a complement of the traditional evaluation methods to improve the architecture design of complex systems. Using the set of elements that compose the architecture design of web applications, an abstraction model is designed with the aim to define the basic structure for the equivalent simulation model. Such a simulation model is detailed using two discrete-event formalisms: Discrete Event System Specification and Routed DEVS. In each case, a mapping is proposed to capture the full structure of the architecture with all its complexity. We compare both solutions to judge their applicability. Hence, the main advantages and disadvantages of both approaches are analyzed with the aim to evaluate their use in the architecture design stage of Software Engineering projects related to web applications.**

***Index Terms*—Software design, Software systems, Systems simulation, Systems architecture.**

I. Introducción

El diseño arquitectónico de un producto de software es un proceso creativo en el que se intenta establecer una organización de los componentes del sistema que satisfaga los requerimientos funcionales y no funcionales propuestos para el mismo [1]. La arquitectura de software comprende los componentes del software, las propiedades visibles de dichos componentes y las relaciones que existen entre ellos [2]. Esta definición no refiere a "buenas" o "malas" arquitecturas, sino que centra su atención en la capacidad del diseño para actuar como vehículo que facilite o dificulte que el producto cumpla con los requerimientos de calidad identificados.

En este contexto, las arquitecturas de software deben ser evaluadas. Este proceso de evaluación no debe centrarse en el uso de una única métrica universal, sino que debe cuantificar atributos individuales y, luego, realizar una compensación de las métricas obtenidas [3]. A fin de evaluar cuantitativamente estos atributos, el desarrollo de métodos y modelos que faciliten la captura de información en relación con las arquitecturas de software constituye un tema de interés para la Ingeniería de Software (IS).

M. J. Blas, INGAR (UTN-CONICET), Avellaneda 3657, Santa Fe, 3000, Argentina (e -mail: mariajuliablas@santafe-conicet.gov.ar).

S. Gonnet, INGAR (UTN-CONICET), Avellaneda 3657, Santa Fe, 3000, Argentina (e -mail: sgonnet@santafe-conicet.gov.ar).

Simular es el proceso de diseñar un modelo de un sistema real y realizar experimentos sobre el mismo a fin de entender su comportamiento y/o evaluar estrategias que mejoren su operación. Esta técnica ha sido ampliamente usada en ramas como la Química, Física y Eléctrica con el objetivo de estudiar el comportamiento de componentes en situaciones definidas y/o de evaluar la capacidad de respuesta de un sistema en un momento dado. También se ha empleado con éxito en el área militar a fin de estudiar, por ejemplo, la utilidad de las estrategias de ataque en situaciones de combate específicas y para entrenamiento [4]. Sin embargo, en la IS la aplicación de esta técnica se ha visto afectada por las propiedades inherentes de los productos que se desarrollan.

Recientemente la IS ha comenzado a usar técnicas de Modelado y Simulación (M&S). Por ejemplo, en [5] los autores hacen uso de un modelo de simulación continuo para mitigar el esfuerzo de estimación de riesgos en proyectos de software. En [6], los autores presentan un esquema general de simulación hibrida que facilita la creación de solicitudes de usuario según diferentes tipos de carga de trabajo. En este contexto, con el objetivo de evaluar atributos de calidad en base a requerimientos no funcionales, y tomando como base el diseño arquitectónico, también se han comenzado a diseñar modelos de análisis basados en M&S. Los autores de [7] hacen uso de simulaciones para evaluar las capacidades IaaS en base a escenarios reales. Por su parte, en [8] se propone un entorno de simulación que posibilita la experimentación con infraestructuras Cloud. Ambos trabajos sientan las bases para el uso de M&S en IS, sin embargo, la obtención de los modelos requeridos no es una tarea sencilla. Su construcción requiere que los arquitectos posean conocimientos específicos de M&S. Aun cuando es posible definir modelos haciendo uso de herramientas de software, el diseño del producto a nivel arquitectónico no suele presentar un grado de estabilidad suficiente para garantizar una única etapa de modelado e implementación. Luego, la introducción de modelos de simulación como estrategia de análisis de calidad implica, además de conocimientos propios de la disciplina. una carga de trabajo adicional a las tareas de diseño arquitectónico.

Normalmente los arquitectos de software hacen uso de *patrones de diseño*. Un *patrón de diseño arquitectónico* define una familia de sistemas en base a un esquema de organización estructural. En este contexto, este trabajo presenta una valoración del formalismo DEVS (Discrete Event System Specification) [9] como soporte al M&S de arquitecturas web basadas en *patrones de diseño*. La particularidad que tienen estas arquitecturas, a diferencia de arquitecturas tradicionales,

https://doi.org/10.1109/TLA.2022.9675459

es que los componentes se definen una única vez durante la etapa de diseño, pero en tiempo de ejecución coexisten múltiples réplicas. Luego, para evaluar su respuesta en ejecución haciendo uso de simulación, es necesario manejar tanto el comportamiento individual de los componentes como así también su coexistencia. Siguiendo los patrones abordados en [10], se presentan dos estrategias de transformación modelo-a-modelo para el M&S basado en DEVS y RDEVS (Routed DEVS) [11]. A partir del diseño de la arquitectura (modelo inicial), se genera un mapeo de cada uno de sus componentes y conexiones con modelos predefinidos. Este proceso da como resultado dos variantes de un mismo modelo, las cuales son analizadas según su contribución y utilidad a la evaluación de arquitecturas. El objetivo del trabajo es evaluar el uso de ambos formalismos en etapa de diseño considerando que el M&S no debe entorpecer la labor de los arquitectos. Entre las principales contribuciones se destacan la propuesta de M&S como complemento a las técnicas de evaluación tradicionales y, además, la valoración de dos formalismos como técnicas para la definición de modelos de simulación.

El resto del trabajo se encuentra estructurado de la siguiente manera. En la Sección II se realiza un análisis de los métodos de evaluación de arquitecturas tradicionales, resumiendo la forma en la cual el uso de simulación puede complementar sus limitaciones. La Sección III analiza el uso de modelos de simulación en IS. La Sección IV introduce los formalismos basados en eventos, enfatizando DEVS y RDEVS como soporte al M&S arquitectónico. Estos fundamentos son usados en la Sección V, la cual presenta su aplicación en arquitecturas de software web. Finalmente, la Sección VI se encuentra destinada a conclusiones y trabajos futuros.

II. Evaluación de Arquitecturas de Software

A. Arquitecturas de Software y Calidad

Según [2], las arquitecturas de software se caracterizan por: *i)* esbozar la estructura organizacional de los componentes del producto, *ii)* definir restricciones de implementación, *iii)* mejorar el estudio del impacto de cambios sobre el producto, *iv)* proporcionar una base para la evaluación del impacto del diseño sobre los atributos de calidad, *v)* facilitar la estimación de las cualidades funcionales del software, *vi)* simplificar la elaboración de planificaciones y estimaciones de costos, y *vii)* contribuir al desarrollo de prototipos evolutivos. En esencia, el diseño arquitectónico se centra en vincular los atributos de calidad (requerimientos no funcionales) con las características funcionales del sistema. Para lograr esta vinculación, el arquitecto selecciona un conjunto de lineamientos que lo ayudan a balancear los distintos aspectos requeridos.

Sin embargo, la característica fundamental del proceso creativo que guía el diseño de arquitecturas de software es que el arquitecto parte de un conjunto de objetivos que no se encuentra claramente definido. La especificación de estos objetivos se obtiene a medida que se lleva a cabo el proceso de solución. Esta particularidad, sumada a la necesidad de ajustar el diseño a los requerimientos (funcionales y no funcionales) del sistema, lleva a que las arquitecturas deban ser evaluadas. Estas evaluaciones deben entenderse como estrategias de verificación que toman las arquitecturas como vehículo para analizar su adecuación a la calidad esperada como resultado del proceso de desarrollo. Luego, alcanzar un nivel de calidad apropiado implica, en todos los casos, lograr una arquitectura adecuada a la naturaleza de los atributos de calidad relevantes.

B. Métodos para la Evaluación de Arquitecturas de Software

Como se ha mencionado, la evaluación del diseño arquitectónico en etapas previas a su implementación es una buena práctica ingenieril. De acuerdo con [3], toda técnica que permita valorar el ajuste de una arquitectura candidata al sistema de software bajo desarrollo es de gran importancia.

Dado el dinamismo con el cual evolucionan los productos de software, recientemente se han comenzado a desarrollar nuevas estrategias de evaluación centradas en la combinación de técnicas tradicionales con técnicas provenientes de otras áreas. En la sección II.B.1 se describen brevemente las principales técnicas de evaluación de arquitecturas, mientras que en la sección II.B.2 se presenta la forma en la cual el uso de simulación (como "nueva estrategia de evaluación") puede complementar las técnicas existentes. Es importante destacar que estas "nuevas estrategias" no buscan reemplazar los métodos ya establecidos; sino que, por el contrario, buscan compensar sus limitaciones. Así, estas "nuevas estrategias" deben verse como complementos de los métodos tradicionales.

1) Métodos Tradicionales

A lo largo de los años, se han definido distintas estrategias para abordar el problema de evaluación de arquitecturas de software. Desde un punto de vista genérico, estas estrategias pueden clasificarse en: *i)* estrategias basadas en escenarios, y *ii)* estrategias basadas en medición cuantitativa o cualitativa.

En el primer caso, los métodos más difundidos son Scenario-based Architecture Analysis Method (SAAM), Architecture Tradeoff Analysis Method (ATAM), y Cost Benefit Analysis Method (CBAM). El método SAAM (análisis basado en escenarios) ha sido propuesto en [12] como una estrategia para la evaluación de arquitecturas en base a escenarios. Estos escenarios se corresponden con evaluaciones cualitativas de la arquitectura. Sin embargo, se ha demostrado que estos escenarios son necesarios, pero no suficientes para predecir y controlar múltiples atributos de calidad. Por este motivo, su uso debe ser complementado con otras técnicas de evaluación (por ejemplo, preguntas orientadas a la cuantificación de los indicadores de calidad prioritarios en cada escenario). En este contexto, surge el método ATAM[1] (análisis de compensación de la arquitectura) como mejora de SAAM. Este método incorpora categorías y árboles de utilidad a los atributos de calidad a ser analizados sobre el diseño de la arquitectura, supliendo así la insuficiencia del método previo. Finalmente, el método CBAM (análisis costo/beneficio) propuesto en [15] se basa en los artefactos producidos por ATAM, modelando los costos y beneficios de las decisiones arquitectónicas utilizadas a fin de obtener una curva definida en términos de

[1] El método ATAM propuesto en [13] corresponde a uno de los métodos de evaluación de arquitecturas de software más estudiados, siendo considerado un método maduro ya que ha sido validado en diferentes dominios [14].

su utilidad/respuesta. De esta forma, este método permite analizar la utilidad de las estrategias/tácticas arquitectónicas según su costo y tiempo de implementación asociado.

En las estrategias de evaluación basadas en medición, es importante destacar que, aunque pueden relevarse mediciones de carácter cuantitativo, la mayoría de las características relevadas a nivel arquitectónico son cualitativas [16]. En este sentido, en la literatura se encuentran técnicas de evaluación para atributos de calidad específicos. Este es el caso de [17], donde se emplean modelos de Procesos de Markov para evaluar la confiabilidad según el estilo arquitectónico usado. Otros enfoques, como por ejemplo el utilizado en [18], han optado por aplicar Teoría de Colas para relevar el rendimiento de la arquitectura (combinando el tamaño de la cola con el tiempo promedio de arribo de peticiones a componentes). Más recientemente, los autores de [19] han usado Teoría de Grafos para analizar la modificabilidad de un diseño arquitectónico. Algunos estudios avanzados han centrado sus esfuerzos en el análisis de más de un atributo de calidad. En estos casos, los autores trabajan con un subconjunto de atributos de calidad cuyo tratamiento puede realizarse de forma conjunta. Por ejemplo, en [20] se usan Redes de Petri para relevar seguridad, confiabilidad y rendimiento sobre un diseño arquitectónico.

Sin embargo, aun cuando los métodos detallados son útiles para la evaluación de arquitecturas, poseen limitaciones. Los enfoques basados en Procesos de Markov requieren, además de la construcción del modelo, su resolución analítica. En conjunto, ambas actividades convierten la tarea de evaluación en una actividad tediosa para los arquitectos [21,22]. Por su parte, si bien la Teoría de Colas facilita la evaluación del rendimiento, no es sencillo utilizar este formalismo para representar otros aspectos de calidad (como seguridad y confiabilidad). De forma similar, las Redes de Petri son útiles únicamente cuando se las utiliza como base para la evaluación de sistemas simples. En sistemas complejos, su resolución se dificulta y, por lo tanto, su aplicación en etapa de diseño ralentiza los tiempos de trabajo. Además, el uso de estas redes requiere simplificar el problema a modelar [20]. Al simplificar, se da lugar a una potencial pérdida de información (asociada a características propias del sistema bajo estudio), lo que puede afectar los resultados finales de la evaluación.

Luego, en términos generales, las limitaciones de los métodos tradicionales se encuentran vinculadas a:

- la imposibilidad de evaluar múltiples atributos de calidad usando un subconjunto reducido de técnicas vinculadas;
- la dificultad asociada a la construcción del modelo requerido para representar el escenario del sistema de software, garantizando un nivel de abstracción adecuado en lo referido a su estructura y características reales;
- las complejidades propias de la técnica de evaluación usada para la construcción y evaluación del modelo que actúa como soporte en la etapa de diseño arquitectónico; y
- el incremento en los tiempos de desarrollo a causa del tiempo asociado a la ejecución del método de evaluación (el cual incluye, además de la evaluación en sí, el diseño, construcción y validación del modelo de soporte).

2) Métodos Complementarios

A fin de complementar tanto las limitaciones de los métodos tradicionales como el estudio de la calidad de los sistemas de software, se han llevado a cabo numerosas investigaciones. Muchos de estos esfuerzos han centrado su atención en el uso de técnicas computacionales para la estimación del grado de adecuación del software final a los requerimientos de calidad. En la mayoría de los casos, los autores proponen el uso de artefactos intermedios combinados con la arquitectura como principales actores del proceso de análisis, tales como [23,24].

Un producto intermedio es cualquier modelo, especificación, documento o código fuente que es creado y utilizado durante el proceso de construcción del sistema de software [16]. De esta manera, la idea detrás de los métodos complementarios es, haciendo uso de un proceso de evaluación predefinido, proveer una predicción razonable de uno o más atributos de calidad del sistema de software objetivo en base a una especificación de calidad unificada. Por ejemplo, en [23] se utiliza la especificación de requerimientos como base para la predicción del atributo rendimiento. Asimismo, en [24] se usa la arquitectura para predecir la confiabilidad del producto resultante, posibilitando ajustar la arquitectura candidata según factores de interés asociados a atributos de calidad.

En la última década, varios autores han usado simulación como complemento de las metodologías de evaluación de arquitecturas tradicionales. Bajo esta perspectiva, se toma el diseño arquitectónico como base para la construcción de un modelo de simulación, por medio del cual se evalúa el comportamiento del software diseñado en base a los atributos de calidad. Estas estrategias han sido aplicadas tanto para arquitecturas de software como para infraestructuras de hardware [7,8,25,26,27]. Luego, la aplicación de M&S como técnica complementaria a las tradicionales de la IS ha dado lugar al surgimiento de nuevas soluciones para problemas preexistentes como ser, por ejemplo, la evaluación de arquitecturas. Una de ellas se presenta en este trabajo en donde, usando formalismos basados en eventos discretos, se analiza el diseño arquitectónico de aplicaciones web.

III. Modelos de Simulación en Ingeniería de Software

Existen múltiples ventajas de emplear simulación en etapas tempranas del proceso de desarrollo de software. Por ejemplo, el uso de un modelo de simulación como soporte en la etapa de diseño posibilita el estudio del impacto de las decisiones arquitectónicas en escenarios artificiales. Un mejor ajuste del diseño en relación con los requerimientos de calidad garantiza una baja presencia de errores y/o costos inesperados en etapas de trabajo futuras. Esto da lugar a sistemas de software robustos y con un alto grado de mantenibilidad. Aún más, los modelos de simulación formulados en la etapa de diseño pueden, luego, ser utilizados como complemento de los componentes de software finales para establecer el mejor curso de acción en tiempo de ejecución (con el objetivo de lograr una correcta adaptabilidad). En este sentido, es evidente que el costo de construir un modelo de simulación y realizar corridas a fin de evaluar su comportamiento, es menor a los costos asociados a la obtención de un producto defectuoso.

Por este motivo, aunque el producto intermedio usado como soporte de la simulación no refleje la totalidad del sistema, el M&S brinda una buena aproximación para analizar la calidad que, eventualmente, tendría el producto final. Cuando esta primitiva se aplica en arquitecturas de software, el beneficio es doble: se estima la calidad del producto final y se evalúa (con una técnica complementaria) la adecuación del diseño.

A. *Abstracciones y Modelos de Software*

Según [28], el término "modelo" refiere a una descripción abstracta de un sistema de interés. Las abstracciones usadas al construir un modelo quedan definidas según el propósito para el cual es construido. Luego, los modelos pueden usarse para representar objetos reales o para describir sistemas en etapas de desarrollo. Un "buen" modelo debe servir como vehículo para llevar a cabo estudios vinculados al sistema final.

En IS se usan diferentes clases de modelos durante el diseño, desarrollo y mantenimiento; en donde la arquitectura de software corresponde a un modelo de diseño. Como se ha enunciado con anterioridad, este modelo se basa en *patrones de diseño*, los cuales proponen estructuras predefinidas para los componentes que dan respuesta a los requerimientos de usuario. Aun así, de forma independiente al uso de patrones, las arquitecturas resultantes del proceso de diseño deben ser evaluadas a fin de determinar su ajuste a la calidad requerida. Al usar M&S para análisis, el modelo de simulación requerido puede derivarse aplicando transformaciones modelo-a-modelo (diseñadas en función de los patrones de diseño del modelo original). Así, partiendo de un modelo de diseño (arquitectura), es posible construir un modelo de abstracción (estructura o escenario de simulación) que, posteriormente, sea traducido a un modelo de simulación formal.

IV. Modelos de Simulación Basados en Eventos

El M&S basado en eventos discretos se caracteriza por operar sobre una base de tiempo continuo, donde el avance de la variable tiempo se lleva a cabo siguiendo intervalos variables. Tales intervalos se obtienen por medio de la planificación de la ocurrencia de eventos a un tiempo futuro. En este contexto, el formalismo DEVS corresponde a un formalismo de M&S basado en la teoría de sistemas que proporciona una metodología general para la construcción jerárquica de modelos de simulación reutilizables de forma modular [9]. Desde su introducción a finales de los 70, este formalismo se ha utilizado como base para la simulación de distintos tipos de sistemas.

El formalismo DEVS describe el comportamiento de un sistema en base a dos niveles de descomposición: descripción de comportamiento (modelo atómico) y descripción de estructura (modelo acoplado). En el nivel más bajo, un modelo atómico describe el comportamiento autónomo de un sistema de eventos discretos como una secuencia de transiciones deterministas entre estados secuenciales junto con la forma en la cual esta secuencia: *i)* reacciona ante eventos de entrada, y *ii)* genera eventos de salida. En el nivel superior, un modelo acoplado describe un sistema como una red de componentes ensamblados (los cuales pueden corresponder tanto a modelos atómicos como a modelos acoplados).

Dada la evolución de los problemas a ser resueltos con técnicas de simulación basadas en eventos discretos, diferentes autores han adaptado el formalismo DEVS para dar respuesta nuevos escenarios. Es decir, al aumentar la complejidad de los problemas (a fin de facilitar su resolución), han surgido nuevos mecanismos de soporte al M&S. El formalismo RDEVS [11] ha sido diseñado como una adaptación de DEVS que busca dar solución a la identificación de eventos en el comportamiento interno de los modelos. Este formalismo actúa como una capa sobre DEVS que provee funcionalidad de ruteo sin necesidad de que el modelador recurra a DEVS para lograr este tipo de funciones [9]. Los componentes básicos de simulación pueden ser modelados en DEVS y, luego, se puede aplicar RDEVS sobre estos modelos para implementar el M&S del ruteo.

El formalismo RDEVS define tres tipos de modelos: esencial, ruteo y red. El modelo esencial refiere a un componente básico de simulación (es decir, una descripción de comportamiento). Sobre este tipo de modelos, se define un modelo de ruteo como una entidad (o nodo) que integra un comportamiento básico con una política de ruteo asociada a los eventos de entrada/salida. Múltiples modelos de ruteo se agrupan en un modelo de red con el objetivo de definir una topología todos-contra-todos que facilite el ruteo de eventos (dejando la decisión de destino a las políticas de ruteo detalladas en los nodos). De esta manera, RDEVS abstrae y organiza el flujo de eventos de forma independiente al comportamiento de los componentes.

A. *Arquitecturas como Modelos basados en Eventos*

Como se ha mencionado con anterioridad, desde un punto de vista estructural las arquitecturas de software esquematizan un conjunto de componentes vinculados por conexiones. Estas conexiones se derivan de la interacción que tiene lugar entre componentes cuando el software busca resolver una solicitud. Así, las conexiones se definen a nivel arquitectónico (en tiempo de diseño) como dependencias entre componentes. Sin embargo, en tiempo de ejecución, no todas las dependencias son utilizadas. En un momento dado, solo las dependencias requeridas para resolver la solicitud actual se encuentran "activas". Es decir, el diseño arquitectónico define todas las dependencias requeridas entre componentes de software. Sin embargo, cuando la arquitectura procesa una solicitud, dicha solicitud únicamente "activa" las dependencias de los componentes de software encargados de procesarla.

Luego, la estructura del modelo de simulación asociado a una arquitectura debe mantener todas las conexiones definidas entre componentes y, además, debe gestionar la distribución de las solicitudes de forma tal que se garantice su correcto procesamiento. Si las solicitudes se modelan como eventos que fluyen a través de los componentes, el problema puede resolverse pensando el M&S como un modelo basado en DEVS que permita administrar la distribución de eventos. En este enfoque, los eventos de entrada representan clases de solicitudes de usuario que "navegan" o "fluyen" sobre los modelos asociados a los componentes arquitectónicos a fin de ser procesadas. Luego, la construcción de modelos de simulación referidos a distintos tipos de arquitecturas puede lograrse tanto con modelos DEVS como RDEVS.

V. Modelos Discretos en Arquitecturas Web

Al construir un modelo de simulación, el modelador debe estudiar tanto el comportamiento del sistema como también su estructura. Partiendo de los elementos que componen el diseño arquitectónico, es posible modelar una abstracción que facilite la representación de ambos niveles en un único modelo.

En [10] se ha presentado una taxonomía de componentes que facilita la representación de arquitecturas de software web. Esta taxonomía identifica 3 tipos de componentes: *componentes de aplicación*, *componentes de administración* y *componentes funcionales*. Los *componentes de aplicación* son elementos usados para especificar el comportamiento de la aplicación web bajo diseño. Estos componentes se clasifican según su dominio en *componente de aplicación definido* (elemento comúnmente usado en una arquitectura web, cuyo comportamiento no varía según el dominio de la aplicación) y *componente de aplicación no definido* (elemento específico referido al dominio de la aplicación web, cuyo comportamiento debe ser especificado por el arquitecto). Los *componentes de administración* refieren a elementos que se utilizan para gestionar automáticamente el comportamiento de la aplicación. Es decir, estos componentes monitorean el funcionamiento de los *componentes de aplicación* de forma predefinida. Así, su comportamiento es bien conocido por el arquitecto y no varía con el dominio de la aplicación. Finalmente, los *componentes funcionales* refieren a un subconjunto de responsabilidades asociadas a funciones elementales. Estos elementos son utilizados para modelar el *comportamiento de componentes de aplicación no definidos*.

La Fig. 1 resume las relaciones entre los diferentes componentes detallando el nivel de abstracción asociado a cada uno de ellos (estructura o comportamiento). Así, los modelos de simulación pueden ser diseñados siguiendo estas abstracciones.

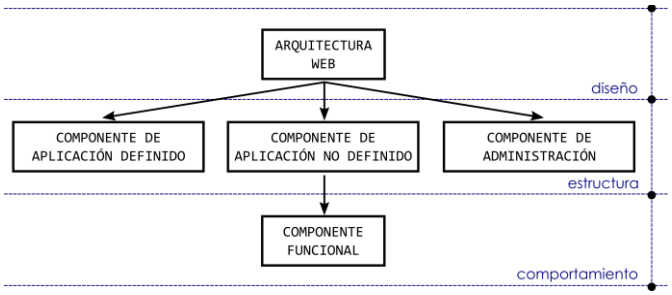

Fig. 1. Componentes arquitectónicos y nivel de referencia.

El modelo de simulación diseñado para dar respuesta a la simulación de una arquitectura de software debe solucionar el problema detallado en la sección IV.A (donde las solicitudes "navegan" por los modelos). La resolución de este problema puede interpretarse como una solución de alto nivel, donde el modelador centra su atención en la estructura del diseño de forma independiente al comportamiento de los componentes (es decir, los componentes son vistos como "cajas negras" que procesan eventos). En este caso, los modelos DEVS y RDEVS propuestos abordan únicamente la definición estructural de la arquitectura. El nivel "comportamiento" queda fuera del alcance de este trabajo. Sin embargo, como se demostrará en la sección V.C, esto no afecta el análisis de los modelos resultantes.

A. Arquitectura de Software Genérica

Comúnmente, las aplicaciones de software web usan un estilo arquitectónico basado en bandas para organizar sus componentes y conexiones. Este estilo brinda una estructura que permite construir aplicaciones flexibles y reutilizables. Para esto, en su forma más simple, define 3 "bandas" (*tiers*): presentación, lógica de negocios y acceso a datos. Mediante esta estructura, el diseño final de la aplicación permite escalar la banda de presentación y el procesamiento informático (lógica de negocios) de forma independientemente al manejo de la información (el cual siempre es más difícil de escalar y, frecuentemente, es contratado a un proveedor de la nube).

La Fig. 2 muestra una arquitectura de 3 bandas diseñada haciendo uso de la herramienta propuesta en [10]. Este ejemplo se corresponde con el caso de estudio "3-tier architecture" propuesto en [29]. Como puede observarse, cada banda se compone de un conjunto de elementos de alto nivel. La banda de presentación (*Presentation Tier*) se compone de un balanceador de carga[2] (*Load Balancer*, componente de aplicación definido) junto con su gestor (*Elastic Load Balancer*, componente de administración), y un componente de aplicación no definido denominado *Presentation*. La banda de lógica de negocios (*Business Logic Tier*) se compone de una cola de mensajes (*Message Queue*, componente de aplicación definido) junto con su gestor (*Elastic Queue*, componente de administración), y un componente de aplicación no definido denominado *Business Logic*. Finalmente, la banda de acceso a datos (*Data Access Tier*) se compone de una cola de mensajes (*Message Queue*, componente de aplicación definido) junto con su gestor (*Elastic Queue*, componente de administración), y un componente de aplicación no definido denominado *Data*.

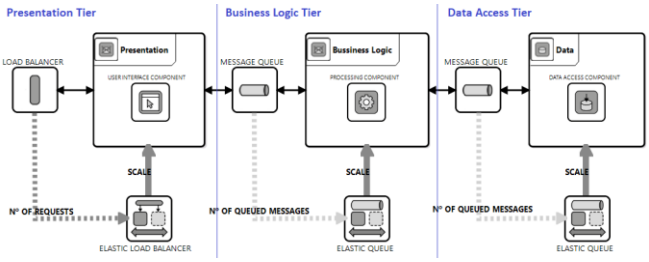

Fig. 2. Arquitectura web de 3 bandas (modelo de diseño).

Los componentes interfaz de usuario (*User Interface Component*), procesamiento (*Processing Component*) y acceso a datos (*Data Access Component*) refieren a componentes funcionales. Estos elementos no forman parte de la estructura del diseño (como se muestra en la Fig. 1). Por ese motivo, su funcionamiento no se analiza en la siguiente sección.

B. Estructura del Modelo de Simulación

La Fig. 3 presenta una abstracción de la arquitectura de software propuesta en la Fig. 2. Este modelo abstracto identifica cada uno de los componentes arquitectónicos de alto nivel (componentes estructurales) a ser representados en el modelo de simulación.

[2] El *Load Balancer* es el intermediario entre las solicitudes y la aplicación.

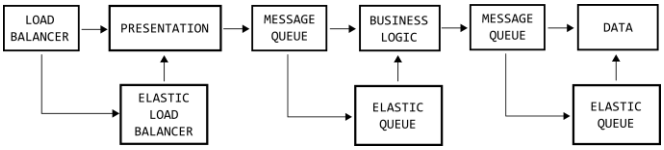

Fig. 3. Estructura de la arquitectura bajo estudio (modelo de abstracción).

Sobre el modelo de abstracción previo, a fin de comparar las capacidades de DEVS y RDEVS para representar esta estructura, se aplica un conjunto de transformaciones a cada tipo de componente. Para DEVS, cada componente abstracto es mapeado a un modelo acoplado siguiendo la estructura definida en la Fig. 4(a). Esta estructura define un componente de alto nivel como un modelo acoplado compuesto de dos modelos atómicos: *Manejador de Eventos* y *Comportamiento*. El *Manejador de Eventos* se encarga de analizar los eventos de entrada según el tipo de solicitud de usuario (paso 1). Si el tipo de solicitud debe ser procesada por el componente de alto nivel asociado al modelo, el *Manejador de Eventos* pasa la solicitud actual al modelo *Comportamiento* (paso 2). Una vez que este finaliza su ejecución, la solicitud es nuevamente enviada al *Manejador de Eventos* para que determine el siguiente destino (paso 3). Al finalizar esta configuración, la solicitud abandona el modelo de simulación acoplado (paso 4).

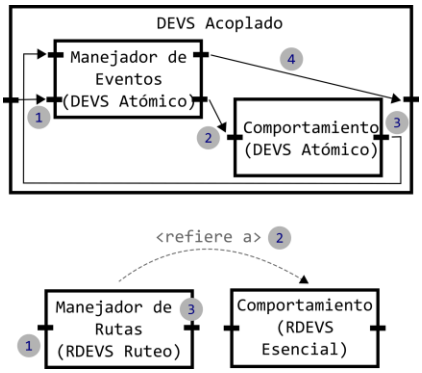

Fig. 4. (a) *Arriba*: Modelo un componente de alto nivel en DEVS. (b) *Abajo*: Modelo un componente de alto nivel en RDEVS.

Para el caso de RDEVS, el esquema de mapeo corresponde a la Fig. 4(b) donde un componente de alto nivel se define como un modelo de ruteo denominado *Manejador de Rutas*. Este modelo refleja el accionar definido en el modelo esencial *Comportamiento* sólo cuando detecta que los eventos de entrada (paso 1) deben procesarse en el componente asociado. Así, el *Manejador de Rutas* "copia" las instrucciones de *Comportamiento* únicamente cuando la solicitud debe ser resuelta (paso 2). Luego de "invocar" a su *Comportamiento*, el *Manejador de Rutas* calcula el próximo destino y propaga la solicitud por su puerto de salida (paso 3).

Las Fig. 5 y 6 esquematizan el mapeo del modelo abstracto de la Fig. 3 a los modelos de simulación finales basados, respectivamente, en DEVS y RDEVS (según la Fig. 4). Como puede observarse, cada componente abstracto es materializado en un modelo independiente. En ambos casos, los elementos sombreados de igual color refieren a elementos de bajo nivel (es decir, al *Comportamiento* de componentes funcionales).

La solución DEVS de la Fig. 5 vincula los componentes en la misma secuencia que el modelo abstracto (Fig. 3). Esto se debe a que el formalismo permite resolver las interacciones definidas a nivel abstracto por medio del uso de acoplamientos entre puertos. Cuando un modelo envía un evento por un puerto de salida, dicho evento es propagado automáticamente por todos los acoplamientos definidos en el puerto. En este contexto, los *Manejadores de Eventos* son usados para determinar si el evento entrante (proveniente de un puerto de salida) debe ser procesado por su *Comportamiento* (ya que, probablemente, no sea el único acoplamiento definido).

Por su parte, la solución propuesta en RDEVS (Fig. 6) trata los vínculos como acoplamientos todos-contra-todos. Esta topología parte de la definición del modelo de red [11] donde, para dejar las tareas de enrutamiento a las políticas de ruteo, los eventos se envían a todos los destinos. Los modelos que definen *Comportamiento* se representan fuera de la red ya que no forman parte de su estructura. Esto permite que distintos modelos compartan un mismo *Comportamiento* (como, por ejemplo, *Message Queue* -MQ- y *Elastic Queue* -EQ-).

C. Esfuerzo de M&S

Aunque no se ha presentado la formalización de los modelos, la representación gráfica usada permite realizar un análisis de las capacidades de cada formalismo al M&S de arquitecturas de software web. En este punto, es importante destacar que los modelos *Comportamiento* son los mismos en ambas soluciones. Mientras que en DEVS estos modelos corresponden a modelos atómicos, en RDEVS corresponden a modelos esenciales. Sin embargo, un modelo esencial es definido como un modelo atómico. Así, el arquitecto modela el comportamiento de los componentes en DEVS de forma independiente a la estrategia de formalización[3].

Como es evidente de la definición estructural, la solución en DEVS emplea una cantidad mayor de modelos que la solución en RDEVS. Un mayor número de modelos implica, implícitamente, un mayor esfuerzo de M&S por parte del arquitecto. Sin embargo, la solución en RDEVS incurre en una mayor cantidad de acoplamientos. Esto, en tiempo de ejecución, puede dar lugar a demoras debido a la constante propagación de todos los eventos de salida hacia los puertos de entrada. Además, la definición estática de las interacciones de componentes por medio de acoplamientos "precableados" (Fig. 5) conlleva a un mayor número de modelos para lidiar con el manejo de eventos. La aplicación de una estrategia dinámica basada en políticas de ruteo flexibiliza la cantidad de modelos, pero requiere un mayor número de acoplamientos.

Luego, puede lograrse un balance entre ambos aspectos si se considera la cantidad de componentes arquitectónicos. En aquellos casos donde las arquitecturas requieran de muchos componentes, una solución basada en RDEVS tendrá un menor esfuerzo de M&S por parte del arquitecto (a expensas de un mayor tiempo de ejecución). Por su parte, en los casos donde la cantidad de componentes sea relativamente baja, una solución basada en DEVS tendrá un mejor rendimiento.

[3] Luego, "RDEVS actúa como una capa sobre DEVS" (ya que los comportamientos se modelan en DEVS y, luego, se usa RDEVS para ruteo).

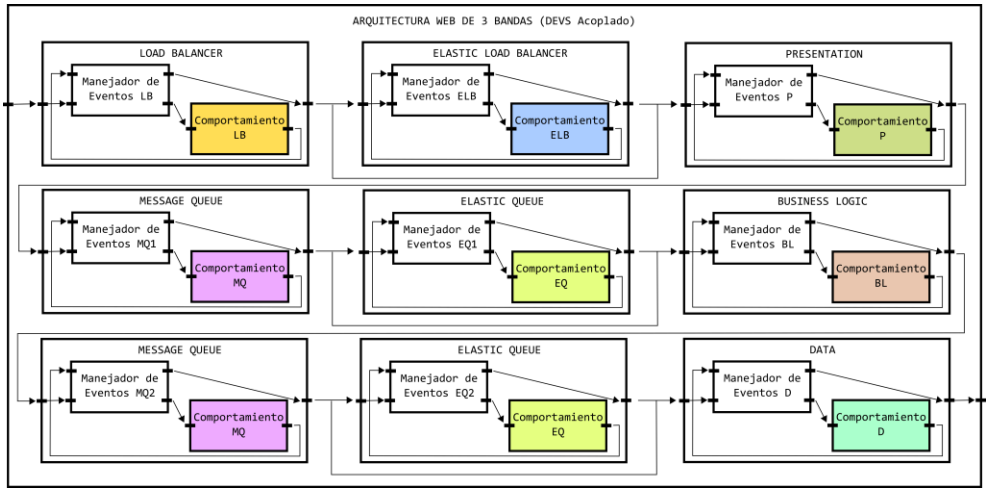

Fig. 5. Modelo de simulación basado en DEVS.

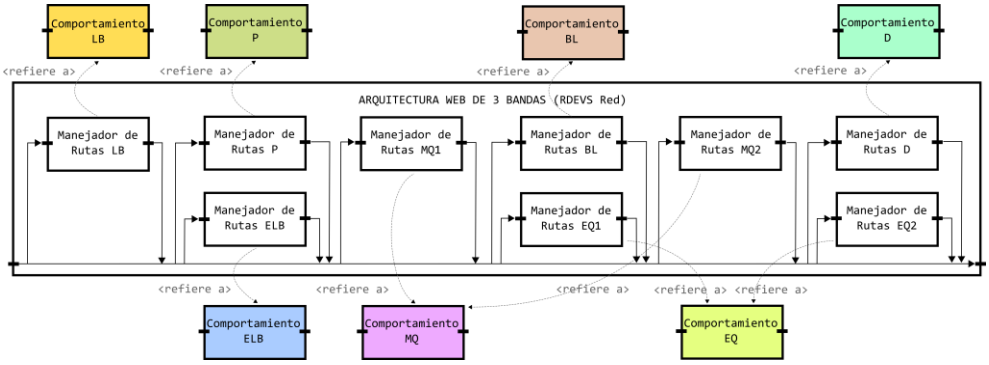

Fig. 6. Modelo de simulación basado en RDEVS.

La capacidad de modificar el modelo resultante también es un factor importante. Las arquitecturas sufren modificaciones a lo largo del ciclo de vida. Luego, es deseable que el modelo de simulación pueda adaptarse de forma simple y sencilla a tales modificaciones. Si un cambio en la arquitectura implica reformular toda su estructura, el enfoque de M&S es costoso. Por otra parte, si el cambio sólo involucra una parametrización del modelo definido, el enfoque es altamente recomendable.

El impacto de un cambio en las interacciones implica un ajuste de todos los modelos involucrados. En DEVS, se deben modificar las interacciones definidas como acoplamientos "precableados" y los *Manejadores de Eventos* asociados. En RDEVS, dado que los modelos se encuentran acoplados entre sí por definición, sólo se deben cambiar las políticas de ruteo.

En ambos casos, la incorporación de un componente conlleva cambios mayores. En DEVS, debe definirse un nuevo modelo y, luego, generar su acoplamiento en la estructura existente. Además, se deben modificar los *Manejadores* de los modelos vinculados. En RDEVS, también debe definirse el modelo, pero su alta en la red requiere únicamente generar los acoplamientos todos-contra-todos. Un cambio en las políticas de ruteo vinculadas también será necesario. Sin embargo, esta tarea solo afecta la parametrización de los modelos existentes. Así, la solución RDEVS facilita la incorporación de cambios ya que lo hace a nivel de parámetros (mientras que la solución DEVS involucra cambios internos en los *Manejadores*).

La Tabla I resume las principales conclusiones alcanzadas, donde la columna 1 refiere a las propiedades que diferencian

TABLA I
COMPARATIVA DEVS Y RDEVS PARA EVALUACIÓN DE ARQUITECTURAS

Propiedad	DEVS	RDEVS
Cantidad de modelos	Mayor cantidad de modelos a diseñar.	Menor cantidad de modelos a diseñar.
Cantidad de acoplamientos	Menor cantidad de "precableados".	Mayor cantidad de todos-contra-todos.
Cambios en interacciones	Ajustes en definición de los modelos.	Modificación de política de ruteo.
Incorporación de componentes	Definición de modelos y ajuste de previos.	Nuevo modelo y parametrización de previos.

DEVS y RDEVS. Como ejemplo de esta comparativa pueden analizarse los modelos propuestos en [27] y [30]. En [27], los modelos corresponden a formalizaciones DEVS, mientras que en [30] se presenta un entorno de M&S basado en RDEVS. Para una misma arquitectura (3 bandas), en [27] se observa una mayor cantidad de modelos más complejos que los detallados en [30]. Luego, [27] y [30] pueden tomarse como pruebas empíricas de los resultados alcanzados en este trabajo.

VI. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se ha presentado el uso de modelos de simulación basados en eventos discretos como alternativa para el estudio de arquitecturas de software web. Tal como se ha detallado, la construcción de este tipo de modelos contribuye a complementar las limitaciones de los métodos tradicionales,

dando lugar a nuevas formas de estudio del desempeño esperado en la aplicación bajo desarrollo. Las estrategias de M&S basadas en enfoques modelo-a-modelo (donde se parte del modelo de diseño, pasando por el modelo de abstracción en el cual cada componente es mapeado a un modelo de simulación según una estructura definida), facilita la obtención de modelos de simulación a bajo costo en etapa de diseño arquitectónico. Se ha analizado la usabilidad de DEVS y RDEVS como soporte al M&S, detallando las características de cada caso. Mientras que los modelos RDEVS mantienen la independencia entre el comportamiento y la estructura de los componentes, los modelos DEVS se adjuntan al escenario dando lugar a soluciones que mejoran el tiempo de ejecución.

La forma en la cual el comportamiento de estos modelos debe ser implementado, es el siguiente paso en esta dirección. Como trabajo futuro, se analizará el impacto de los componentes funcionales en la definición de los modelos atómicos y esenciales de las soluciones DEVS y RDEVS, respectivamente. Además, se espera aplicar la propuesta en diferentes tipos de escenarios como ser, por ejemplo, patrones de microservicios y arquitecturas orientadas a servicios.

REFERENCIAS

[1] I. Sommerville, *Software engineering*, 10th ed. Pearson, 2018.

[2] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Boston, MA, USA: Addison-Wesley Professional, 2012.

[3] M. Barbacci, M. Klein, T. Longstaff, and C. Weinstock, "Quality attributes," SEI, Carnegie-Mellon University, Pittsburg, PA, USA, 1995.

[4] A. Bruzzone, and M. Massei, "Simulation-based military training", in *Guide to Simulation-Based Disciplines*, 2017, pp. 315-361.

[5] J. Gomes, J. Montenegro, J. Dos Santos, J. Barbosa, and C. Costa, "A Strategy Using Continuous Simulation to Mitigate Effort Estimation Risks in Software Projects", *IEEE Latin America Transactions*, vol. 17, no. 8, pp. 1390-1398, 2019, DOI: 10.1109/TLA.2019.8932373.

[6] M. Blas, S. Gonnet, and H. Leone, "Modeling User Temporal Behaviors Using Hybrid Simulation Models", *IEEE Latin America Transactions*, vol. 15, no. 1, pp. 341-348, 2017, DOI: 10.1109/TLA.2017.7854631.

[7] G. Fenner, A. Lima, J. De Souza, J. Moura, and T. Bezerra, "Supporting Infrastructure as a Service Capacity Management through Business Scenarios Simulation," *IEEE Latin America Transactions*, vol. 18, no. 03, pp. 473-486, 03/2020, DOI: 10.1109/TLA.2020.9082718.

[8] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms", *Software: Practice and Experience*, vol. 41, no. 1, pp. 23-50, 2011, DOI: 10.1002/spe.995.

[9] B. Zeigler, A. Muzy, and E. Kofman, *Theory of modeling and simulation: discrete event & iterative system computational foundations*, 3er ed. Academic Press, 2018.

[10] M.J. Blas, H. Leone, and S. Gonnet, "Modelado y verificación de patrones de diseño de arquitectura de software para entornos de computación en la nube," *Revista Ibérica de Sistemas e Tecnologias de Informação*, no. 35, pp. 1-17, 12/2019, DOI: 10.17013/risti.35.1-17.

[11] M.J. Blas, S. Gonnet, and H. Leone, "Routing structure over discrete event system specification: a DEVS adaptation to develop smart routing in simulation models," in WSC, Las Vegas, NV, USA, 2017, pp. 774-785, DOI: 10.1109/WSC.2017.8247831.

[12] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *IEEE software*, vol. 13, no. 6, pp. 47-55, 1996, DOI: 10.1109/52.542294.

[13] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in ICECCS, Monterey, USA, 1998, pp. 68-78, DOI: 10.1109/ICECCS.1998.706657.

[14] M. Babar, and I. Gorton, "Comparison of scenario-based software architecture evaluation methods,", in 11th Asia-Pacific Software Engineering Conference, Busan, South Korea, 2004, pp. 600-607, DOI: 10.1109/APSEC.2004.38.

[15] R. Kazman, J. Asundi, and M. Klien, "Making architecture design decisions," SEI, Carnegie-Mellon University, Pittsburg, PA, USA, CMU/SEI-2002-TR-035, 2002.

[16] S.T. Albin, *The art of software architecture: design methods and techniques*, Hoboken: John Wiley & Sons, 2003.

[17] W. Wang, Y. Wu, and M. Chen, "An architecture-based software reliability model," in PRDC, Hong Kong, 1999, pp. 143-150, DOI: 10.1109/PRDC.1999.816223.

[18] B. Spitznagel, and D. Garlan, "Architecture-based performance analysis," in SEKE, 1998, pp. 146-151.

[19] F. Bachmann, L. Bass, M. Klein, and C. Shelton, "Experience using an expert system to assist an architect in designing for modifiability," in WICSA, 2004, pp. 281-284, DOI: 10.1109/WICSA.2004.1310710.

[20] K. Fukuzawa, and M. Saeki, "Evaluating software architectures by coloured Petri nets," in SEKE, 2002, pp. 263-270, DOI: 10.1145/568760.568807.

[21] A. Immonen, and E. Niemelä. "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software & Systems Modeling*, vol. 7, no. 1, 49, 2008, DOI: 10.1007/s10270-006-0040-x.

[22] L.K. Singh, A.K. Tripathi, and G. Vinod, "Software reliability early prediction in architectural design phase: overview and limitations," *Journal of Software Engineering and Applications*, vol. 4, no. 3, pp. 181-186, 2011, DOI: 10.4236/jsea.2011.43020.

[23] J. Dargan, E. Campos-Nanez, P. Fomin, and J. Wasek, "Predicting systems performance through requirements quality attributes model," *Procedia Computer Science*, vol. 28, no. 1, pp. 347-353, 2014, DOI: 10.1016/j.procs.2014.03.043.

[24] R. Roshandel, N. Medvidovic, and L. Golubchik, "A Bayesian model for predicting reliability of software systems at the architectural level," in QoSa, 2007, pp. 108-126, DOI: 10.5555/1784860.1784871.

[25] H. Koziolek, S. Becker, R. Reussner, and J. Happe, "Evaluating performance of software architecture models with the Palladio component model," in *Software Applications: Concepts, Methodologies, and Tools*, P. Tiako, Ed. Information Science Reference, 2009, ch. 64, pp. 1111-1134, DOI: 10.4018/978-1-60566-006-6.ch005.

[26] A. Meiappane, B. Chithra, and P. Venkataesan, "Evaluation of software architecture quality attribute for an internet banking system," *International Journal of Computer Applications*, vol. 62, no. 19, pp. 21–24, 2013, DOI: 10.5120/10189-5062.

[27] M.J. Blas, S. Gonnet, and H. Leone, "Building simulation models to evaluate web application architectures," in CLEI, 2016, pp. 1-12, DOI: 10.1109/CLEI.2016.7833339.

[28] C. Nielsen, P. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: Basic concepts, model-based techniques, and research directions," *ACM Computing Surveys*, vol. 48, no. 2, 18, 2015, DOI: 10.1145/2794381.

[29] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*, Germany: Springer Science & Business Media, 2014.

[30] M. Blas, H. Leone, and S. Gonnet, "Modeling and simulation framework for quality estimation of web applications through architecture evaluation", SN Applied Sciences, vol. 2, no. 374, 2020, DOI: 10.1007/s42452-020-2171-z.

Maria J. Blas is a Postdoctoral Fellow at INGAR and an Assistant Professor in the Information Systems Department at UTN. She received her PhD degree in Engineering from UTN in 2019. Her research interests include discrete-event M&S.

Silvio Gonnet received his PhD degree in Engineering from UNL in 2003. He currently holds a Researcher position at CONICET. His research interests are models to support design processes, software engineering and conceptual modeling.