A DEVS Based Methodological Framework for Reinforcement Learning Agent Training

Ezequiel Beccaria, Verónica Bogado, Jorge A. Palombarini, *UTN, CIT Villa María*

***Resumen*—Reinforcement Learning has become one of the fastest growing fields of artificial intelligence due to the successful application of its techniques into several domains. In this way, the integration of intelligent agents based on Reinforcement Learning into information systems is a current reality. However, the way in which they "learn" requires a simulation model of the process that must be controlled to obtain large volumes of risk-free information. In this work, a methodological framework to support the training of Reinforcement Learning agents using DEVS is proposed. This framework provides the steps and elements required to implement RL Agents with the purpose of accelerating the agent learning and reducing training costs. Also, it allows modeling the dynamics of complex systems in a modular and hierarchical way, favoring the reuse of simulation components, since it is based on DEVS formalims fundamentals.**

***Index Terms*—Support for Training RL Agents, Reinforcement Learning, DEVS, AI-enabled systems**

I. Introducción

La Inteligencia Artificial (IA) es, actualmente, uno de los campos que presentan mayor cantidad de desafíos dentro de las nuevas tecnologías de la información, ya que cada vez se encuentra con mayor frecuencia presente en la sociedad en la forma de smart houses, BuildTech, asistentes virtuales de procesos de decisiones, etc. Particularmente, en la última década, el área del Aprendizaje por Refuerzos (Reinforcement Learning - RL, [1]) ha alcanzado una serie de avances sin precedentes que han permitido resolver problemas de decisión de gran complejidad. Este éxito, se debe en gran parte al empleo de RL como mecanismo para resolver problemas modelados como Procesos de Decisión de Markov (Markov Decision Process - MDP [2]), como por ejemplo: Jugar juegos de Atari® [3], [4], [5], jugar Go [6], [7], controlar robots [8], controlar videojuegos complejos [9] o generar planes de producción y manufactura inteligente en industrias 4.0 [10].

A nivel conceptual, el RL propone la existencia de una entidad denominada agente (agente RL), que aprende a partir de la interacción con un entorno o ambiente; dicho entorno normalmente representa un proceso, o sistema, que el agente debe controlar empleando un conjunto predeterminado de acciones de control. Como resultado del aprendizaje, el agente genera una política de selección de acciones (o política de control) que le permite, luego de realizado el entrenamiento, controlar el entorno de manera satisfactoria en tiempo real. Para aprender una política de control, un agente RL requiere un gran volumen de experiencia durante el proceso de entrenamiento. Esto es, necesita experimentar las acciones de control sobre el entorno que controla, para determinar cuáles de ellas son las indicadas en diferentes estados del mismo. Dicha información, es provista como realimentación al agente RL por medio de una señal de refuerzo (recompensa) que consiste en un número real, mediante la cuál el agente determina que tan buena, o no, fue la acción realizada previamente [1].

Desde la perspectiva de RL, estos problemas de control pueden variar, en gran medida, en base a la dificultad que poseen para que un agente inteligente pueda generar una política de control óptima. Aquellos problemas que presentan mayor grado de complejidad son los que cuentan con un gran espacio de estados, un gran espacio de acciones, donde se da la ocurrencia de eventos exógenos (evento aleatorio que afecta el estado del proceso y que el agente no puede predecir), donde el tiempo en el que se ejecuta la acción de control es parte de la definición del estado del proceso, y las acciones de control se toman en intervalos de tiempo irregulares [11]. En este tipo de problemas, los eventos exógenos hacen que se pierda la deseada "propiedad de Markov". Esta propiedad asegura que conociendo un estado s_0 y ejecutando una acción a, el proceso transicionará a un nuevo estado s_1 con una probabilidad predecible.

En este sentido, la naturaleza del aprendizaje de un agente RL requiere de un período de entrenamiento donde dicho agente debe experimentar (e.g. observar el efecto) de las distintas acciones sobre el proceso o sistema que debe aprender a controlar. Esto implica experimentar tanto las acciones óptimas (mayor recompensa), como así también las sub-óptimas (menor recompensa). Sin embargo, la mencionada situación hace imposible llevar a cabo el entrenamiento de un agente RL en entornos de alto riesgo, donde experimentar una acción no óptima puede implicar pérdidas económicas o poner en riesgo vidas humanas.

En este contexto, es necesario contar con un modelo de simulación que se comporte de manera similar al proceso original a controlar, con el objetivo de reducir el tiempo de aprendizaje y permitir al agente RL experimentar distintas acciones en cada uno de los posibles estados del proceso, sin enfrentar los riesgos de realizar el entrenamiento sobre el proceso real. Muchas de las industrias que cuentan con procesos que pueden controlarse automáticamente, donde es viable la aplicación de RL, no cuentan con un modelo de simulación disponible del mismo. Por lo cual, existe la necesidad de modelar la dinámica (e.g. cómo el proceso controlado

Ezequiel Beccaria es becario doctoral de la Facultad Regional Villa María - UTN, Villa María, X5900 HLR Argentina, e-mail: ebeccaria@frvm.utn.edu.ar.

Verónica Bogado y Jorge A. Palombarini son profesores-investigadores de la Facultad Regional Villa María - UTN, Villa María, X5900 HLR Argentina y el CIT Villa María - CONICET-UNVM, Villa María, Córdoba, X5900 HLR Argentina.

transiciona de un estado a otro) de cada proceso o sistema antes de realizar el entrenamiento de un agente RL.

Esta situación, presenta la necesidad de contar con un marco formal que defina la metodología y los elementos requeridos para poder generar modelos del ambiente de operación de los procesos de control necesarios, para llevar a cabo el entrenamiento de agentes RL de manera sistematizada y repetible, con el objetivo principal de reducir riesgos y costos. Al favorecer el uso de metodologías y herramientas formales para la definición de los mencionados modelos, se facilita la verificación y/o validación formal del problema de control, acercándolo a la realidad y reduciendo la ocurrencia de errores.

En este trabajo, se propone un framework metodológico para entrenar agentes RL de forma sistemática con el objetivo de representar problemas de control donde existan eventos exógenos dependientes del tiempo. Es decir, captura no solo la dinámica del proceso a controlar, sino también, los eventos externos que influyen sobre este. Para ello, se propone el uso de Procesos de Decisión Generalizados Semi-Markovianos (Generalized Semi-Markov Decision Process - GSMDP [12]) como formalismo para el modelado del problema de control. Un GSMDP no solo contempla la evolución del estado del proceso a través de las acciones ejecutadas por el agente, sino que además, habilita la ocurrencia de eventos exógenos no controlables que modifican el estado del proceso sin intervención alguna del agente. Por otro lado, se propone el uso de DEVS (Discrete EVent System Specification [13]) como motor de simulación que brinda soporte al entrenamiento del agente RL. DEVS permite representar diferentes tipos de formalismos como: Autómatas de estado finito, Redes de Petri, Diagrama de Estados, entre otros [14], [15]. Adicionalmente, utilizar DEVS como herramienta para simular el problema de control, presenta como ventajas: la utilización del mismo formalismo para representar el modelo GSMDP y su simulación; al ser pensado a partir de la teoría general de sistemas (TGS), insta a la construcción de modelos jerárquicos; facilita la modularidad y reutilización; y representa los procesos de control formalmente, pudiendo someterse, en caso de ser necesario, a técnicas de chequeo de modelos para verificar el funcionamiento de los mismos [16], [17].

Para validar el presente trabajo, se aplica el framework metodológico propuesto para entrenar 2 agentes RL como solución al problema control *The Subway Problem* [18]. En este problema, el agente es un administrador que decide cuántos trenes se van a encontrar en operación en una línea de metro, controlando el proceso al habilitar o deshabilitar la operación de cada uno de los trenes del mismo, con el objetivo de maximizar el balance entre el costo de operación de la línea y el ingreso por venta de pasajes. Las arquitecturas de agentes RL elegidas para el desarrollo del caso de estudio son DDQN [19] y A3C [5], las cuales constituyen el punto de partida de todos los actuales avances en el área.

Este trabajo está estructurado de la siguiente manera: La Sección II describe los trabajos relacionados; En la Sección III, se define el problema que resuelve la propuesta; en la Sección IV, se presenta en detalle el framework metodológico propuesto, la Sección V introduce un caso de estudio donde se aplica y valida la propuesta, la Sección VI discute ventajas y desventajas del uso de la metodología propuesta y, finalmente; en la Sección VII, se detallan las conclusiones y trabajos futuros.

II. Trabajos Relacionados

Desde la perspectiva de la comunidad de simulación de procesos, en [20] se presenta DEVS Markov Model (DMM) como una especialización de DEVS capaz de representar "Procesos de Markov", compuestos de sub-modelos más simples, los cuales poseen sus respectivos estados y transiciones. Sin embargo, el foco de este trabajo deja afuera la interacción de estos modelos con un agente controlador al no contemplar procesos de tipo MDP. La problemática de modelar y simular un MDP utilizando DEVS es abordada en [21], en donde se propone un framework genérico para el modelado y simulación basado en eventos discretos de MDPs jerárquicos, utilizando RL para obtener una política de control sobre el proceso. En este caso, si bien se contempla el modelado y simulación de problemas de control, no considera procesos más generales (como los GSMDPs), limitando el uso del framework a procesos que pueden ser modelados como un MDP.

Desde la perspectiva del área de RL, en [18] se presenta un método para la resolución de GSMDPs utilizando una variación "on-line" de programación dinámica llamada iteración de política [1]. Si bien este trabajo presenta una aproximación interesante en cuanto al modelado y simulación de GSMDPs utilizando DEVS, el trabajo no presenta metodologías para la solución de este tipo de problemas.

Otro esfuerzo realizado en el área es el proyecto OpenAI© Gym [22]. Este proyecto presenta una serie de herramientas destinadas a reducir la complejidad y el tiempo necesario para la puesta en marcha de un agente RL, como así también, su comparación con otros agentes en una serie de entornos de control pre-definidos. Adicionalmente, permite la implementación de entornos de control personalizados, manteniendo una única interfaz de interacción entre el agente y su entorno, facilitando la utilización de un mismo agente en diferentes entornos de control con una mínima cantidad de cambios. Pero, como contrapartida, al no estar basado en un formalismo, carece de todas las bondades que el uso de un lenguaje formal brinda a la hora de verificar el correcto funcionamiento de los entornos implementados con esta herramienta.

III. Definición del Problema y solución General

Al utilizar RL para dar solución a un problema de control, es necesario contar con un modelo de simulación del ambiente de operación en el que se encuentra el agente, en donde llevar a cabo el entrenamiento de este de forma rápida y segura. Esto es, reducir el tiempo de entrenamiento al realizar el mismo en un modelo simulado, como así también experimentar todo tipo de acciones (buenas y malas) para poder observar sus consecuencias a largo plazo sin correr el riesgo de experimentar acciones sub-óptimas en un proceso de control real. Al modelar problemas de control complejos, donde las transiciones entre estados se ven influenciadas, no solo por las acciones realizadas por el agente RL, sino también por la ocurrencia de eventos exógenos dependientes del tiempo,

que complejizan aun más el proceso de aprendizaje de una política en dichos problemas, que no pueden modelarse usando MDPs clásicos. Así, surge el problema de reducir costos al modelar y entrenar agentes RL en ambientes de operación con las características mencionadas, siguiendo una metodología sistematizada y formal, para solucionar el problema de control real de forma precisa.

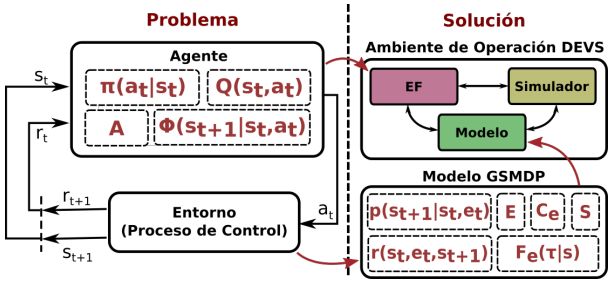


Fig. 1: Representación gráfica del Entorno de Entrenamiento de agentes RL basado en DEVS.

En este trabajo y en el marco de la metodología propuesta, se plantea el uso de GSMDP como formalismo para realizar el modelado del problema de decisión en donde ocurren eventos exógenos dependientes del tiempo. En la Fig. 1, se definen gráficamente los aspectos involucrados en el problema de decisión y cómo estos se representan en la solución propuesta. Así, el *Entorno* se captura en un *Modelo GSMDP* que describe el funcionamiento del proceso de control a simular: el conjunto S de estados, un conjunto E de eventos, los relojes de activación C_e de los eventos E, la función transición $P(s_{t+1} \mid s_t, e_t)$, la función de recompensa $r(s_t, e_t, s_{t+1})$ y la función $F_e(\tau \mid s)$ encargada de determinar el momento de activación para cada evento e en el estado s.

Una vez definido el modelo del entorno, es necesario poder ejecutar una simulación del mismo para que el agente pueda experimentar acciones y observar los efectos de las mismas en términos de transiciones de estados y recompensas recibidas. Al utilizar el formalismo DEVS como motor de simulación del entorno o proceso bajo control, se debe realizar una transformación previa del *Modelo GSMDP* definido, a un modelo DEVS (*Modelo*). Para esto, se utilizan las reglas de transformación definidas en [11]. El *Agente* integra el mecanismo de decisión y aprendizaje, siendo embebido en otro modelo DEVS incluido dentro del marco experimental (*EF*). Finalmente, en este trabajo se emplea DEVSJAVA [23] como motor de simulación del modelo DEVS (*Simulador*).

IV. Framework Metodológico para Entrenamiento de Agentes RL

El Framework Metodológico propuesto se representa en la Fig. 2 mediante un diagrama de actividad (UML). En la misma, se capturan los elementos (actividades, entradas, salidas, productos, herramientas) necesarios para aplicar el framework con el fin de entrenar agentes RL utilizando DEVS. A su vez, se definen las actividades que se deben realizar, el orden de las mismas y los roles responsables de llevar a cabo cada una de las actividades. Por último, se definen los objetos o información requerida para cada actividad, como así también, los productos o resultados de salida de estas.

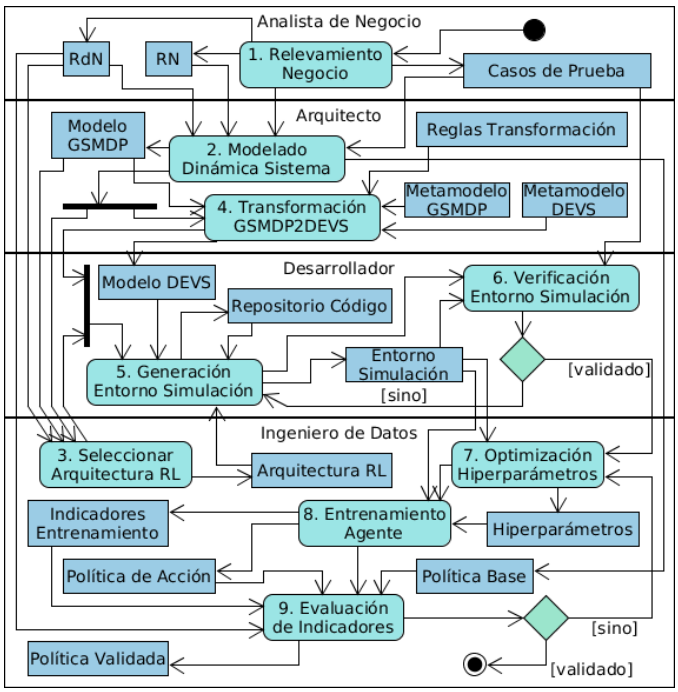


Fig. 2: Framework metodológico para entrenamiento de agentes RL basado en DEVS.

Los roles definidos se han adaptado en base a [24]. En la Tabla I, se enumeran y detallan los distintos roles que intervienen en la metodología, definiendo así las partes involucradas necesarias para entrenar un agente RL con modelos DEVS.

Tabla I: Roles de la metodología.

Rol	Descripción
Analista de Negocio	Es el encargado de realizar el relevamiento necesario para entender los requisitos y el funcionamiento del proceso de control que se quiere simular.
Arquitecto	Es el encargado de definir el modelo de simulación del proceso de control basándose en lo relevado por el rol Analista, teniendo en cuenta la infraestructura tecnológica necesaria para desarrollar la actividad.
Desarrollador	Se encuentra a cargo de la generación (código ejecutable) del entorno de simulación que será utilizado para entrenar el agente inteligente.
Ingeniero de Datos	Es el encargado de la configuración del agente, de la ejecución de las simulaciones y de evaluar el rendimiento de la política de control encontrada por el agente inteligente.

El punto de inicio en el framework metodológico propuesto es la actividad "1. Relevamiento Negocio". En esta actividad, desarrollada por el rol *Analista de Negocio*, se realiza un análisis sobre el problema de control, generando los requerimientos de negocio (RN) sobre el comportamiento del entorno de simulación y las restricciones de negocio (RdN) sobre la posible implementación del mismo (e.g. restricciones en el tiempo de respuesta del agente RL, precisión de la solución, etc.). También, se definen los casos de prueba que el futuro entorno de simulación deberá superar luego de ser implementado.

La siguiente actividad es “2. Modelado Dinámica Sistema”, la cual es llevada a cabo por el rol *Arquitecto*. En esta, se realiza la formulación del problema desde el punto de vista de la dinámica del proceso a controlar. Con esta información, se realiza la representación del proceso de control como un modelo GSMDP con sus respectivos componentes.

A partir de este punto, las siguientes dos actividades son realizadas de forma simultanea. La primera de ellas es “3. Seleccionar Arquitectura RL” realizada por el rol *Ingeniero de Datos*. Esta actividad implica el análisis del modelo GSMDP definido previamente, los RN y los RdN, para determinar qué arquitectura de agente RL permite cumplir con los requerimientos establecidos, seleccionando la misma para ser utilizada.

Por otro lado, en la actividad “4. Transformación GSMDP2DEVS” realizada por el rol *Arquitecto*. Una vez definido el modelo del entorno, es necesario poder ejecutar una simulación del mismo para que el agente pueda experimentar acciones y observar los efectos de las mismas en términos de transiciones de estados y recompensas recibidas. Al utilizar el formalismo DEVS como motor de simulación del entorno, se debe realizar una transformación previa del *Modelo GSMDP* definido, a un modelo DEVS (Ver Fig. 1). Para ello, se emplea el “Metamodelo DEVS” y el “Metamodelo GSMDP” predefinidos para esta actividad, los cuales incluyen conceptos necesarios para llevar a cabo la transformación. El *Agente* integra el mecanismo de decisión y aprendizaje, siendo embebido en otro modelo DEVS incluido dentro del marco experimental (*EF*).

Una vez definido el “Modelo DEVS” y la “Arquitectura RL” a utilizar, el resto de las actividades del framework propuesto se realizan de forma secuencial. La siguiente actividad es “5. Generación Entorno Simulación”, en la cual, el rol *Desarrollador* se centra en la implementación del entorno de entrenamiento, el cual se compone del modelo de simulación DEVS definido en la actividad previa, como así también de la implementación de la arquitectura elegida para el agente RL, en un lenguaje de programación que permita la simulación del modelo que representa al proceso controlado. La siguiente actividad “6. Verificación Entorno Simulación”, también llevada a cabo por el rol *Desarrollador*, es donde se ejecutan todos los casos de prueba definidos sobre el entorno de simulación para verificar el correcto comportamiento del mismo.

Una vez verificado el “Entorno de Simulación”, las restantes actividades son realizadas por el rol *Ingeniero de Datos*. En la actividad “7. Optimización Hiperparámetros”, se realiza una búsqueda exhaustiva en el espacio de hiperparámetros del agente, para determinar el conjunto de los mismos que logre una mejor política de control sobre el entorno.

En la actividad “8. Entrenamiento Agente”, el agente implementado experimenta repetitivamente distintas situaciones simuladas del proceso controlado con el fin de aprender una política de selección de acciones (“Política de Acción”) sobre el mismo. Durante este proceso, se recolectan medidas de rendimiento que son utilizadas para determinar si la política aprendida por el agente evoluciona favorablemente durante el entrenamiento. Y finalmente, en la actividad “9. Evaluación de Indicadores”, el rendimiento de la política aprendida por el agente es evaluado. Se compara el rendimiento de la política aprendida contra otra política de acción predefinida utilizada como base de rendimiento (“Política Base”), provista a priori, por un experto en el proceso de control. En caso de no superar el rendimiento de la política base, la metodología propone volver a la actividad “7. Optimización Hiperparámetros”.

Las actividades, las entradas necesarias para realizarlas, las salidas que se generan una vez ejecutada cada actividad, así como las técnicas y herramientas utilizadas para desarrollar las actividades del framework propuesto se detallan en las Tablas II, III y IV, V, para los roles Analista de Negocio, Arquitecto, Ingeniero de Datos y Desarrollador respectivamente.

Tabla II: Actividades del rol Analista de Negocio.

Actividad 1: Relevamiento Negocio
Salidas: RN (Requerimientos de Negocio - Característica dinámica del proceso y ambiente de operación); RdN (Restricciones de Negocio - Condiciones sobre el proceso a controlar, ya sean físicas, lógicas, etc); Casos de Prueba (Conjunto de escenarios en los cuales debe verificarse el funcionamiento del modelo de simulación desarrollado.)
Técnicas y herramientas: Observación, Tormenta de ideas, Prototipado, Ingeniería Inversa, Entrevistas, Cuestionarios, etc.

Tabla III: Actividades del rol Ingeniero de Datos.

Actividad 3: Seleccionar Arquitectura RL
Entradas: RdN y Modelo GSMDP (Modelo que describe el comportamiento del proceso que se requiere controlar).
Salidas: Arquitectura RL (Arquitectura elegida para desempeñar la tarea de control sobre el proceso); Política Base (Política de acción provista por un experto utilizada como base de rendimiento para evaluar la política de acción aprendida por el agente)
Técnicas y herramientas: Evaluación de distintas arquitecturas de algoritmos RL (DQN [4], A3C [5], PPO [25], etc.). Esta actividad depende mayoritariamente de la experiencia del Ingeniero de Datos en la resolución de este tipo de problemas.
Actividad 7: Optimización Hiperparámetros
Entradas: Entorno Simulación.
Salidas: Hiperparámetros (Conjunto de hiperparámetros óptimos que permiten configurar el comportamiento del agente al momento de aprender una política de acción durante el entrenamiento).
Técnicas y herramientas: Búsqueda Aleatoria [26], Algoritmos Genéticos [27], Optimización Bayesiana [28], entre otros.
Actividad 8: Entrenamiento Agente
Entradas: Hiperparámetros y Entorno Simulación.
Salidas: Indicadores Entrenamiento (Medidas de rendimiento del agente que sirven para determinar qué tan buena es la política aprendida), Política de Acción (Política de selección de acciones sobre el proceso controlado aprendida por el agente).
Técnicas y herramientas: El objetivo del agente es maximizar la recompensa total G obtenida en cada episodio de entrenamiento $G = \sum_t^T r_t(s, e, s')$.
Actividad 9: Evaluación Indicadores
Entradas: Indicadores Entrenamiento, RdN, Política de Acción y Política Base.
Salidas: Política de Acción Validada (Política de acción que cumple con los requisitos de negocio establecidos).
Técnicas y herramientas: Se compara el rendimiento de la política aprendida contra otra política de acción predefinida utilizada como base de rendimiento, provista a priori, por un experto en el proceso de control.

Tabla IV: Actividades del rol Desarrollador.

Actividad 5: Generación Entorno Simulación
Entradas: Arquitectura RL, Modelo DEVS, Repositorio Código (Código que puede ser reutilizado durante la implementación del modelo de simulación).
Salidas: Entorno Simulación (incluye el Modelo de simulación ejecutable basado en DEVS del proceso controlado, el simulador, marco experimental y el agente RL con su respectiva arquitectura), Repositorio Código.
Técnicas y herramientas: Desarrollo basado en reuso de componentes.
Actividad 6: Verificación Entorno Entrenamiento
Entradas: Entorno Simulación, Casos de Prueba.
Técnicas y herramientas: Chequeo de modelos [16], [17], Pruebas de Unidad [29], etc.

Tabla V: Actividades del rol Arquitecto.

Actividad 2: Modelado Dinámica Sistema
Entradas: RN, RdN.
Salidas: Modelo GSMDP, Política Base.
Técnicas y herramientas: El proceso es representado como un GSMDP.
Actividad 4: Transformación GSMDP2DEVS
Entradas: Modelo GSMDP, Metamodelo GSMDP (Captura los conceptos del lenguaje GSMDP), Metamodelo DEVS (Captura los conceptos básicos del lenguaje DEVS [30]), Reglas de Transformación (Especifica el conjunto de reglas requeridas para transformar los elementos de un GSMDP a un modelo DEVS).
Salidas: Modelo DEVS (Instancia particular de un modelo DEVS para la instancia del modelo GSMDP del proceso de control).
Técnicas y herramientas: El "modelo GSMDP" es transformado a una instancia de un modelo formal DEVS siguiendo las "Reglas de Transformación". Para ello, se emplea el "Metamodelo DEVS" y el "Metamodelo GSMDP" pre-definidos para esta actividad, los cuales incluyen conceptos necesarios para llevar a cabo la transformación.

V. Caso de estudio

Como caso de estudio para validar el framework metodológico propuesto, se ha implementado un problema de control de la literatura de agentes conocido como *The subway problem* descrito en [18]. A continuación, se detallan las tareas realizadas en cada una de las actividades definidas según la metodología propuesta.

V-1. Relevamiento Negocio: En este problema de control, el agente es un administrador que decide cuántos trenes se van a encontrar en operación en una línea de metro, controlando el proceso al habilitar o deshabilitar la operación de cada uno de los trenes, con el objetivo de maximizar el balance entre el costo de operación de la línea y el ingreso por venta de pasajes. Las entidades definidas en el problema de control son: los pasajeros, las estaciones donde esperan los pasajeros y los trenes que los transportan.

El problema de control cuenta con 4 tipos diferentes de eventos:

- El administrador decide poner en operación un nuevo tren.
- El administrador decide quitar de operación uno de los trenes que está en funcionamiento.
- El arribo de un nuevo pasajero a una de las estaciones.
- El arribo de uno de los trenes que se encuentran en operación a una estación.

Con respecto a las *Restricciones de Negocio (RdN)*, fueron definidas las siguientes:

- Puede haber un máximo de 4 trenes en operación de manera simultanea.
- Los Trenes pueden llevar un máximo de 50 pasajeros.
- Pueden esperar para abordar un tren, un máximo de 50 pasajeros en cada estación.
- Se puede dar la ocurrencia simultanea de dos o mas eventos.

V-2. Modelado Dinámica Sistema: En esta actividad, se procede a definir el problema como un GSMDP. Un GSMDP es definido como una tupla dada por $\langle S, E, P, r\rangle$, donde S es el conjunto de estados definidos en la Tabla VI.

Tabla VI: Espacio de Estados en The subway problem.

Variable	Rango de Valores	Descripción
ns_i	$\{0, 50\} \subset \mathbb{N}$	Número de cola finita de pasajeros esperando en la estación $i \mid i \in \{0, \ldots, 5\} \subset \mathbb{N}$.
f_i	$\{0, 1\} \subset \mathbb{N}$	Banderas que indican si la estación i está ocupada por un tren.
nt_i	$\{0, 50\} \subset \mathbb{N}$	Número de pasajeros a bordo del tren $j \mid j \in \{0, \ldots, 3\} \subset \mathbb{N}$.
p_j	$\{-1, \ldots, 5\} \subset \mathbb{Z}$	Posición actual de tren j.
t	$\{0, \ldots, 1440\} \subset \mathbb{R}^+$	Tiempo del día en minutos.

El conjunto de eventos E que pueden ocurrir en el problema de control pueden visualizarse en la Tabla VII. Algunos de estos son eventos controlables, acciones que pueden ser ejecutadas por el agente, y otros no controlables o exógenos.

Tabla VII: Eventos en The subway problem

Evento	Exógeno?	Descripción
ap_i	Si	Pasajero arribando a la estación i siguiendo una distribución normal de probabilidad
mt_j	Si	Arribo de tren j a una estación. Este evento incrementa el valor de la variable de estado p_j y hace $f_pj = 1$. También varía el número de pasajeros a bordo del tren y los pasajeros en la estación p_j.
ad_j	No	Activa al tren j, el cual comienza a funcionar en la estación 1. Esta acción solo es válida si la estación 1 se encuentra libre ($f_1 = 0$).
re_j	No	Enviar al tren j al garaje, lo cual implica que el tren deja de funcionar. Esta acción solo es válida en el caso de que el tren se encuentre en la estación cero ($p_j = 0$).
a_∞	No	Acción "No Operar" (NOP). Esta acción implica que el agente no realiza ninguna acción sobre el entorno y el mismo evoluciona al siguiente estado a partir de los eventos exógenos generados en el mismo entorno.

P es la función de transición, la cual, puede ser definida por partes. A modo de ejemplo, la Ecuación (1) describe el comportamiento del proceso cuando un nuevo pasajero arriba a la estación i.

$$P(s_1 \mid s_0, ap_i) = \begin{cases} ns_i + 1 & \text{si } ns_i < 50 \\ ns_i & \text{sino} \end{cases} \quad (1)$$

Por último, se tiene la función de recompensa que es descrita en la Ecuación (2). La misma esta compuesta de tres

términos que describen la recompensa obtenida por el agente ante cada uno de los posibles eventos del modelo. El término rp modela el pago de los tickets, cada vez que un pasajero arriba a una estación, el término rs modela la penalidad recibida por el agente cada vez que un tren es activado y por último, el término rt modela el costo de funcionamiento de un tren cada vez que este arriba a una estación, multiplicado por la cantidad tiempo que estuvo en funcionamiento desde la última transición de estado del problema.

$$r(s,t,e,s',t') = rp(e) + rs(e) + \sum_{j=0}^{3} rt(t,t') \quad (2)$$

donde

$$rp(e) = \begin{cases} 1{,}5 & \text{si } e = ap_i \\ 0{,}0 & \text{sino} \end{cases}$$

$$rs(e) = \begin{cases} -2{,}0 & \text{si } e = ad_j \\ 0{,}0 & \text{sino} \end{cases}$$

$$rt(t,t') = \begin{cases} \int_{t=0}^{t'} -2{,}0t\ dt & p_j <> -1 \\ 0{,}0 & \text{sino} \end{cases}$$

V-3. Seleccionar Arquitectura RL: En esta actividad se opto por utilizar dos alternativas que sean suficientemente representativas como para demostrar el funcionamiento de marco metodológico propuesto. Particularmente, se trabajó con Deep RL, siendo elegidas las arquitecturas DDQN [19] y A3C [5], las cuales constituyen el punto de partida de todos los actuales avances en el área. Cada una de las arquitecturas elegidas posee una red neuronal artificial de tipo feedforward como predictor de la función de valor y/o política del agente.

V-4. Transformación GSMDP2DEVS: La transformación del modelo GSMDP a DEVS se realiza siguiendo las reglas de transformación definidas en [18]. De esta forma, cada evento no controlable definido en la Tabla VII, será descrito por un modelo DEVS atómico encargado de modelar el reloj de activación de cada evento exógeno denominado *EventGen* –ver Fig. 3–.

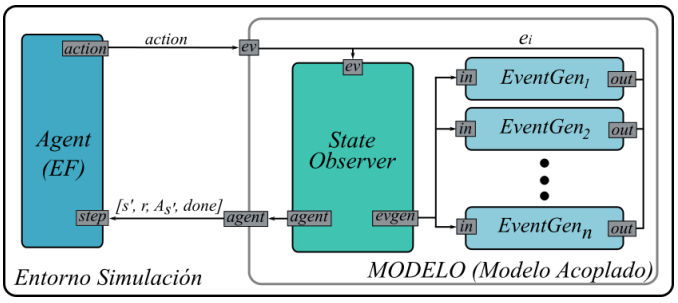


Fig. 3: Conexiones entre los diferentes modelos DEVS utilizados para representar un GSMDP.

Para cada una de las estaciones (seis en total), habrá una instancia del modelo atómico *EventGen* que modela los arribos de los pasajeros a las estaciones, siguiendo una distribución normal con media variable exponencialmente a lo largo del entrenamiento del agente y desviación estándar constante. Con respecto al arribo de los trenes activos, por cada tren (cuatro en total) existe también una instancia del modelo *EventGen* que determina el tiempo de arribo del tren correspondiente a la siguiente estación, siguiendo una distribución normal constante a lo largo del entrenamiento del agente.

Por otro lado, existe un modelo DEVS atómico denominado *StateObserver* (Expresión (3)), encargado de mantener el estado global del proceso y de manejar el arribo de los distintos eventos que alteren el mismo, incluyendo las acciones ejecutadas por el agente RL (ver Fig. 3). Por ende, cada vez que se dispara un evento exógeno, o una acción ejecutada por el agente, el modelo *StateObserver* es el encargado de alterar el estado actual del ambiente simulado, activar/desactivar las distintas instancias del modelo *EventGen*, notificar al agente del nuevo estado s' del ambiente simulado y de las acciones disponibles que este puede realizar en s'. Para ello, este modelo posee un puerto de entrada "ev" donde recibe los eventos que se activaron en cada instante. Además, cuenta con dos puertos de salida: "ev_gen" es utilizado para activar/desactivar las instancias del modelo *EventGen*, y el otro "agent", es utilizado para transmitir el estado actual del modelo luego de una transición. El estado S está compuesto por la fase en la que se encuentra la instancia del modelo DEVS (P), el estado del proceso controlado representado por un vector real de n elementos $\mathbb{R}^n$, la recompensa acumulada hasta la ultima transición externa (r) dada por $rf(s, Event, s', e)$ y el tiempo restante hasta la próxima transición interna de la instancia $\mathbb{R}_0^+$. La dinámica está dada por las transiciones externas que permiten transicionar al modelo a un nuevo estado del proceso GSMDP que representa a través de la función $p(s, Event)$, luego de lo cual, se dispara una transición interna, y por ende, la función de salida envía al puerto "agent" el estado de modelo GSMDP representado. Finalmente, el modelo se vuelve "pasivo" hasta que ocurra una nueva transición externa.

$$StateObserver = (X, Y, S, ta, \delta_{ext}, \delta_{int}, , \delta_{con}, \lambda) \quad (3)$$

donde

$$InPorts = \{\text{“}ev''\};\ X_{ev} = Event$$
$$X = \{(p,v) \mid p \in InPorts, v \in X_p\}$$
$$OutPorts = \{\text{“}ev\_gen'', \text{“}agent''\}$$
$$Y_{ev\_gen} = \mathbb{R}_0^+ \times \{true, false\}$$
$$Actions = \{a_0, a_1, \cdots, a_n\}$$
$$Y_{agent} = \mathbb{R}^n \times \mathbb{R} \times \{true, false\} \times \{a \mid a \subseteq A\}$$
$$Y = \{(p,v) \mid p \in OutPorts, v \in Y_p\}$$
$$P = \{\text{“}passive'', \text{“}active''\};\ ta(s) \leftarrow \infty$$
$$S = P \times \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}_0^+$$
$$s_0 = (\text{“}passive'', s_{init}, 0, \sigma)$$
$$p(s, Event) = \mathbb{R}^n$$
$$done(s) = \{true, false\}$$
$$rf(s, Event, s', e) = \mathbb{R}$$
$$activeEvents(s, done) = \mathbb{R}_0^+ \times \{true, false\}$$
$$enabledActions(s) = \{a \mid a \subseteq A\}$$
$$\delta_{ext}(\text{“}passive'', s, r, \sigma, e, (\text{“}ev'', v)) = (\text{“}active'', p(s,v),$$
$$rf(s,\ v,\ p(s,v),\ e), 0)$$
$$\delta_{int}(\text{“}active'', s, r, \sigma) = (\text{“}passive'', s, 0, \infty)$$
$$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$$
$$\lambda(\text{“}active'', s, r, \sigma) = (\text{“}ev\_gen'', activeEvents(s, done(s)))$$
$$(\text{“}agent'', \{s, r, done(s), enabledActions(s)\})$$

El *Modelo* (ver Fig. 3), representa el ambiente con el que el agente interactúa. El mismo, es un modelo DEVS acoplado

conformado por N instancias del modelo *EventGen* y una instancia del modelo *StateObserver*.

Por último, el modelo atómico *Agent* es donde se implementa la funcionalidad específica de cada arquitectura de agente RL. Este modelo permanece pasivo hasta que transiciona externamente δ_{ext} recibiendo al puerto "step" información del estado actual del entorno con el cual interactúa el agente. En base al estimulo recibido, el modelo responde con la acción de control elegida a través del puerto "action". Por limitaciones de espacio, solo se define formalmente el modelo *StateObserver*.

V-5. Generación Entorno Simulación: Debido a los requerimientos de negocio (RdN) definidos previamente por el rol analista, se lleva a cabo la programación del entorno de simulación utilizando el lenguaje de programación Java, utilizando la librería DEVSJAVA [23] como motor de simulación del modelo del entorno. La librería "DeepLearning4j" [31] fue utilizada para la implementación y entrenamiento de los algoritmos RL.

V-6. Verificación Entorno Entrenamiento: Una vez finalizado el desarrollo y basándose en los casos de prueba definidos, se procede a realizar las pruebas pertinentes para verificar el correcto funcionamiento del entorno de simulación. En la Tabla VIII, se definen 2 casos de prueba a modo de ejemplo para los eventos ap_i y mt_i definidos en la Tabla VII. En los mismos, se define el comportamiento que debe presentar el modelo de simulación en cada uno de los escenarios.

Tabla VIII: Casos de Prueba en "The subway problem"

Rol	Descripción
Arribo de pasajero a la estación i.	Si $ns_i < 50 \implies ns_i = ns_i + 1$ $rp(s, e, s') = 1{,}5$
Arribo de tren a la estación i. bp_j: Pasajeros que bajan del tren j. lp_j: Lugares libres en el tren j. sp_j: Pasajeros que suben al tren j.	Si $f_i = 0$: $f_i = 1;\ p_j = i$ $bp_i \sim \mathcal{N}(b\mu_i, b\sigma_i)nt_j$ $lp_j = nt_i - bp_j$ $sp_j = \begin{cases} lp_j & \text{si } ns_i \geq lp_j \\ ns_i & \text{sino} \end{cases}$ $nt_i = nt_i - bp_j + sp_j;\ ns_i = ns_i - sp_j$

V-7. Optimización Hiperparámetros: Para la optimización de los hiperparámetros de las arquitecturas de agentes seleccionadas, se procedió a realizar un tuneo de los mismos utilizando, por su simplicidad y eficiencia, búsqueda aleatoria [26]. En la Tabla IX se pueden visualizar los parámetros encontrados para cada una de ellas.

Tabla IX: Hiperparámetros de los agentes RL en el Entorno "The subway problem"

Hiper-Parámetro	DDQN	A3C
γ	0,8460	0,59
α	0,0206	$15e-5$
C (DDQN) / t_{max} (A3C)	88	40
Batch Size	256	-
Delta de Decisión (Tiempo máximo entre acción)	27	27

V-8. Entrenamiento Agente: Se procede a realizar el entrenamiento de los agentes. El mismo consta de cinco experimentos por agente, de dos mil episodios de entrenamiento cada uno. Todos los experimentos se llevaron a cabo en una computadora con un procesador AMD Ryzen© 7 1800 3.60 GHz x 8, con 32 GB of RAM DDR4 corriendo en un sistema operativo Ubuntu 18.04 64 bits. El tiempo requerido para llevar a cabo el entrenamiento de los agentes fue de 94 minutos para la arquitectura DQN y de 45 minutos para la arquitectura A3C.

V-9. Evaluación Indicadores: En esa actividad, se analizan las medidas recolectadas para determinar si la política de selección de acciones encontrada por los agentes es satisfactoria. Para ello, se evalúa la recompensa promedio obtenida por cada uno de los agentes elegidos a lo largo del entrenamiento, promediando la recompensa obtenida por cada agente en cada uno de los experimentos realizados por cada episodio y se calcula la desviación estándar de la misma a lo largo de los entrenamientos.

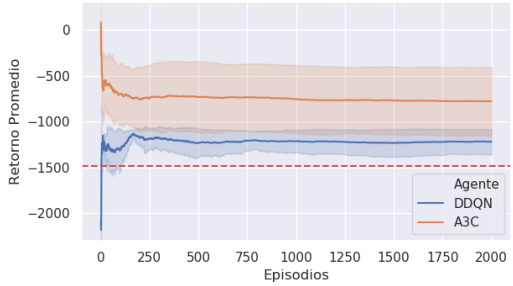


Fig. 4: Retorno promedio por episodio obtenido por los agentes entrenados empleando la metodología propuesta. La linea de puntos representa la performance de la *Política Base.*

En la Fig. 4 se puede ver la evolución de cada uno de los agentes durante el entrenamiento con respecto a la política de base (Linea de puntos). En este caso, la política de base utilizada implica mantener a lo largo de todo el episodio 2 trenes en funcionamiento. Claramente, la arquitectura A3C es la que logro una media de rendimiento superior a lo largo del entrenamiento, pero como contrapartida, posee una mayor desviación estándar que la arquitectura DDQN. Por otra lado, ambas arquitecturas han logrado superar el desempeño de la política de base, lo cual implica que mas allá del mejor rendimiento de una arquitectura por sobre la otra, ambas han aprendido una política de control que se desempeña mejor que una política fija definida a priori.

Al analizar el comportamiento de la curva de aprendizaje de la arquitectura A3C, la misma no es correcta debido a que esta no se estabiliza cerca de la recompensa máxima obtenida. Por esto, se optó por volver a la actividad "7.Optimización Hiperparámetros" y realizar una nueva búsqueda de los hiperparámetros para la arquitectura A3C. Una vez definido un nuevo conjunto de hiperparámetros para la arquitectura ($\gamma = 0{,}94$ y $\alpha = 1 - e5$), se volvió a realizar el entrenamiento de la misma obteniendo los resultados que se muestran en la Fig. 5.

Por ultimo, si comparamos el desempeño de los agentes en los últimos 10 episodios de entrenamiento (ver Tabla X), donde se estima que la política ya no cambia en gran medida, se puede ver que la tendencia vista a lo largo del entrenamiento se mantiene y que la arquitectura A3C se destaca por sobre la arquitectura DDQN.

Con respecto a la política de control aprendida por el agente *A3C*, se selecciono la política de control aprendida del mejor

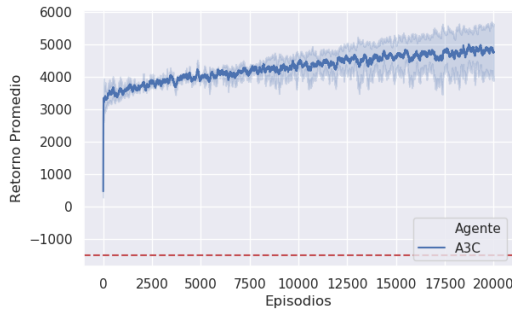


Fig. 5: Retorno promedio por episodio obtenido en el segundo entrenamiento de la arquitectura A3C.

Tabla X: Retorno promedio de cada agente durante los últimos 10 episodios de entrenamiento comparados con la *Política Base*.

Agente	Promedio	Mejor	Peor
A3C	**5652.39**	**6433.11**	**4768.45**
DDQN	-1196.43	-348.3	-1491.78
Política Base	-1481.13	-1475.45	-1485.41

experimento realizado sobre esta arquitectura y se analizó el comportamiento de la misma en diferentes estados claves del problema de control (ver Fig. 6). En el primer ejemplo, cuando las estaciones se encuentran abarrotadas de pasajeros esperando para abordar un tren, la política aprendida define que la acción de control que mayor retorno futuro va a proveer es poner un nuevo tren en operación (Acción *ST*). Ahora, en el segundo caso, cuando solamente hay un tren en operación y las estaciones cuentan con pocos pasajeros, la política aprendida es no realizar acción alguna (Acción *NOP*). Por ultimo, si hay mas de un tren en funcionamiento (dos en este caso), y solo algunos pasajeros esperando en las estaciones, la política aprendida define que sacar de operación uno de los trenes es la mejor acción de control (Acción *GT*).

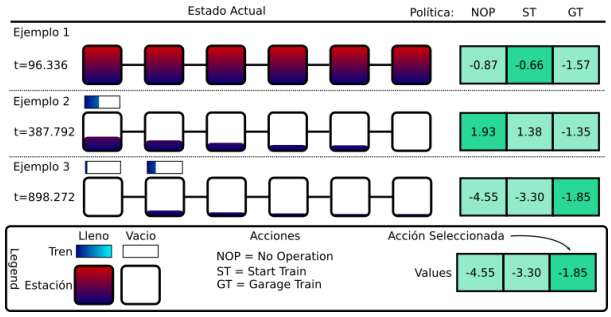


Fig. 6: Política aprendida por la arquitectura A3C para tres estados seleccionados.

VI. Discusión

El uso de agentes RL para el control de procesos industriales requiere la utilización de modelos de simulación de dichos procesos para llevar a cabo el proceso de entrenamiento de dichos agentes. Al formular políticas de control para un proceso industrial de alto riesgo, el modelado y la simulación de estos procesos mediante el uso de herramientas formales se convierte en algo indispensable para evitar la incertidumbre, los riesgos asociados y las posibles pérdidas económicas de una mala correlación entre el proceso real y el simulado. Además, utilizar un framework metodológico y contar con la definición de los pasos, actividades, herramientas y técnicas, entradas y salidas para entrenar agentes RL, permite acelerar dicho proceso al facilitar los medios para su implementación, favoreciendo además su replicabilidad.

Si comparamos la metodología contra el uso de OpenAI© Gym [22], al no ser una metodología, OpenAI© Gym no cuenta con las actividades, roles y elementos requeridos definidos claramente, dificultando al usuario el proceso de entrenamiento. Además, el framework propuesto, al estar basado en un formalismo (DEVS), en el que se incluye el modelado y la implementación del problema de control, posibilita el uso de técnicas para la verificación formal de los modelos de simulación, permitiendo una mayor precisión derivada de una mejor representación de la realidad. Por ultimo, los modelos DEVS son genéricos y aplicables a cualquier problema. Esto, hace posible el uso del framework metodológico, a usuarios que no conozcan el formalismo DEVS, ya que este, se encuentra oculto en la librería desarrollada. El usuario, solo tiene que ajustar los parámetros de los modelos DEVS definidos para representar el modelo GSMDP deseado. Esta ventaja toma mas fuerza, cuando se encuentre automatizada la transformación de modelos entre GSMDP y DEVS.

Como contrapartida, el uso de la metodología propuesta es un proceso costoso, ya que requiere un modelado formal del proceso de control. Al momento de implementar procesos de control simples, esta metodología presenta un costo excesivo, haciendo más conveniente el uso de herramientas mas directas, como es el caso de OpenAI© Gym.

VII. Conclusiones y Trabajo Futuro

La IA presenta grandes desafíos a nivel conceptual y tecnológico, provenientes de la complejidad inherente a la integración de sus técnicas y metodologías en sistemas software, lo que requiere de herramientas que brinden soporte durante su desarrollo. Particularmente, el empleo de RL en un amplio y variado conjunto de problemas procedentes de diferentes dominios requiere poder entrenar los agentes de forma cada vez más eficiente, en el sentido de reducir tiempos y costos. Así, el framework propuesto permite la implementación del proceso de entrenamiento de agentes RL de forma estructurada y con herramientas formales, permitiendo generar una política de control sobre los procesos mediante experimentación con una realidad simulada, reduciendo riesgos a la hora de implementar entornos de entrenamiento para agentes RL en problemas de control de escala industrial. En este sentido, DEVS provee ventajas para capturar la dinámica del entorno con el que interactúa el agente RL, ya que, al basarse en la TGS proporciona herramientas para la construcción jerárquica de modelos que involucren tanta complejidad como se requiera en pos de emular con precisión la realidad del proceso que se debe controlar.

Como trabajo futuro, se prevé automatizar varias de las actividades del framework propuesto. Los procesos de transformación de modelos se automatizarán empleando Model

driven-Development, Model-driven Architecture y los estándares relacionados. Además, se desarrollarán las interfaces gráficas para la interacción con el arquitecto, desarrollador e ingeniero de datos y las interfaces entre componentes.

REFERENCIAS

[1] "Sutton & Barto Book: Reinforcement Learning: An Introduction," http://incompleteideas.net/book/the-book-2nd.html, 2018.

[2] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Aug. 2014.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv:1602.01783 [cs]*, Feb. 2016.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971 [cs, stat]*, Sep. 2015.

[9] P. Zhu, X. Li, P. Poupart, and G. Miao, "On Improving Deep Reinforcement Learning for POMDPs," *arXiv:1704.07978 [cs]*, Apr. 2017.

[10] D. Preuveneers and E. Ilie-Zudor, "The intelligent industry of the future: A survey on emerging trends, research challenges and opportunities in Industry 4.0," *Journal of Ambient Intelligence and Smart Environments*, vol. 9, no. 3, pp. 287–298, Jan. 2017.

[11] E. Rachelson, "Temporal markov decision problems," Ph.D. dissertation, Université Paris 6, 2009.

[12] H. L. S. Younes and R. G. Simmons, "Solving Generalized Semi-Markov Decision Processes Using Continuous Phase-Type Distributions," p. 6, 2004.

[13] B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press, 2018.

[14] S. Borland and H. Vangheluwe, "Transforming statecharts to DEVS," *Summer Computer Simulation Conference (Student Workshop)*, pp. S154–S159, 2003.

[15] N. Giambiasi, J.-L. Paillet, and F. Châne, "Simulation and verification II: From timed automata to DEVS models," in *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation*. Winter Simulation Conference, Jul. 2003, pp. 923–931.

[16] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model Checking: Algorithmic Verification and Debugging," *Commun. ACM*, vol. 52, no. 11, pp. 74–84, Nov. 2009.

[17] M. H. Hwang and B. P. Zeigler, "A Modular Verification Framework Based on Finite & Deterministic DEVS," p. 9, 2006.

[18] E. Rachelson, G. Quesnel, F. Garcia, and P. Fabiani, "A Simulation-based Approach for Solving Generalized Semi-Markov Decision Processes," *Frontiers in Artificial Intelligence and Applications*, pp. 583–587, 2008.

[19] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *arXiv:1509.06461 [cs]*, Sep. 2015.

[20] C. Seo, B. P. Zeigler, and D. Kim, "DEVS Markov Modeling and Simulation: Formal Definition and Implementation," in *Proceedings of the Theory of Modeling and Simulation Symposium*, ser. TMS '18. San Diego, CA, USA: Society for Computer Simulation International, 2018, pp. 1:1–1:12.

[21] C. Kessler, L. Capocchi, J. Santucci, and B. Zeigler, "Hierarchical Markov decision process based on DEVS formalism," in *2017 Winter Simulation Conference (WSC)*, Dec. 2017, pp. 1001–1012.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[23] H. S. Sarjoughian and B. Zeigler, "Devsjava: Basis for a devs-based collaborative m&s environment," *Simulation Series*, vol. 30, pp. 29–36, 1998.

[24] A. De Mauro, M. Greco, M. Grimaldi, and P. Ritala, "Human resources for big data professions: A systematic classification of job roles and required skill sets," *Information Processing & Management*, vol. 54, no. 5, pp. 807–817, 2018.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," p. 12, 2017.

[26] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

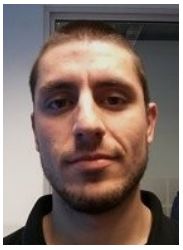
[27] J. R. Koza, *Genetic Programming*, 1997.

[28] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2951–2959.

[29] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[30] A. C. H. Chow and B. P. Zeigler, "Parallel DEVS: A parallel, hierarchical, modular modeling formalism," in *Proceedings of Winter Simulation Conference*, Dec. 1994, pp. 716–722.

[31] D. Team *et al.*, "Deeplearning4j: Open-source distributed deep learning for the jvm," *Apache Software Foundation License*, vol. 2, 2016.



Ezequiel Beccaria received the Engineering degree in Information Systems Engineering from the Universidad Tecnológica Nacional of Argentina (UTN-Argentina) in 2010. Is an aspiring PhD. in Information Systems at the Universidad Tecnológica Nacional and has developed several works in the area of reinforcement learning and deep learning.

Veronica Bogado Verónica Bogado received a PhD degree in Engineering with Information Systems Engineering (2013) from the Universidad Tecnológica Nacional, Facultad Regional Santa Fe. She is currently working at the Department of Information Systems Engineering of the Facultad Regional Villa María, Universidad Tecnológica Nacional. Her current research interests are related to software quality evaluation, software architecture design, and M&S of complex systems, particularly DEVS formalism and its application to software problems.

Jorge A. Palombarini received his PhD. in Information Systems Engineering from the Universidad Tecnológica Nacional of Argentina (UTN-Argentina) in 2014. Current academic position includes Associate Professor of Artificial Intelligence and Syntax and Semantic of Languages in the UTN, and Assistant Research Fellow of CONICET. He was a software developer in private sector institutions and auditor of information systems on Universidad Nacional de Villa María, Argentina (UNVM-Argentina). His current research interest includes Reinforcement learning, Deep Learning, Cognitive systems and Formal Frameworks for industrial process modeling and simulation.