

Un Framework Basado en DEVS para la Integración de Agentes de Aprendizaje por Refuerzo en Entornos de Simulación de Eventos Discretos

A DEVS-Based Framework for Integrating Reinforcement Learning Agents into Discrete Event Simulation Environments

Doctorando:

Ezequiel BECCARIA

<https://orcid.org/0000-0002-3770-9107>

Facultad Regional Villa María, Universidad Tecnológica Nacional, Villa María, Córdoba, Argentina
ezequielbeccaria@gmail.com

Director:

Jorge Andrés PALOMBARINI

Facultad Regional Villa María, Universidad Tecnológica Nacional, Villa María, Córdoba, Argentina

Codirectora:

Verónica BOGADO

Instituto Multidisciplinario de Investigación y Transferencia Agroalimentaria y Biotecnológica, Consejo Nacional de Investigaciones Científicas y Técnicas – Universidad Nacional de Villa María, Villa María, Córdoba, Argentina

Resumen

La simulación de eventos discretos requiere cada vez más agentes inteligentes de toma de decisiones, pero no existe un enfoque formal estandarizado dentro del formalismo de Especificación de Sistemas de Eventos Discretos (DEVS) para incorporarlos. Se propone un framework que provee *building blocks* DEVS reutilizables y formalmente especificados para construir entornos de simulación —modelados formalmente como Procesos de Decisión Semi-Markovianos Generalizados (GSMDP)— que exponen interfaces estandarizadas para agentes de Aprendizaje por Refuerzo (RL). Se definen tres modelos componibles (*StateObserver*, *EventGenerator* y *Agent*) que encapsulan el protocolo de interacción con el agente de RL, permitiendo que el modelado específico del dominio siga siendo el foco central. La propuesta se valida en un caso real de replanificación ferroviaria, donde tres algoritmos de RL fundamentalmente distintos se evalúan sobre el mismo entorno DEVS sin modificarlo.

Palabras clave: DEVS; problemas de decisión secuencial; entornos de simulación integrados con RL; building blocks DEVS reutilizables; GSMDP

Abstract

Discrete event simulation increasingly requires intelligent decision-making agents, yet no standardized formal approach exists for incorporating them within the Discrete Event System Specification (DEVS) formalism. We propose a framework providing reusable, formally specified DEVS building blocks for constructing simulation environments, formally modeled as Generalized Semi-Markov Decision Processes (GSMDP), exposing standardized interfaces for Reinforcement Learning (RL) agents. Three composable models are defined (*StateObserver*, *EventGenerator* and *Agent*) that encapsulate the RL interaction protocol, allowing domain-specific modeling to remain the central focus. The approach is validated on a real-world railway rescheduling study, where three fundamentally different RL algorithms are evaluated on the same DEVS environment without modifying it.

Keywords: DEVS; sequential decision problems; RL-integrated simulation environments; reusable DEVS building blocks; GSMDP

Introducción

Discrete Event System Specification (DEVS; Zeigler et al., 2019) es un framework formal consolidado para el modelado y la simulación de sistemas complejos, aplicado en dominios diversos como la evaluación de arquitecturas de software (Bogado et al., 2017) y el modelado de procesos de ruteo (Blas et al., 2022). Incorporarle capacidades inteligentes de decisión extiende su aplicabilidad a *Problemas de Decisión Secuencial* (SDP). El Aprendizaje por Refuerzo (RL; Arulkumaran et al., 2017) es hoy la metodología de referencia para aprender políticas de control en este tipo de problemas, con resultados notables en dominios como los juegos (van Hasselt et al., 2016), lo que generó una demanda creciente de modelos DEVS capaces de interactuar con agentes de RL sin perder sus garantías formales ni sus ventajas composicionales. Sin embargo, el formalismo carece de componentes estandarizados para exponerlos a dichos agentes: no hay mecanismos formales para recibir acciones, observar el estado, señalar recompensas ni gestionar episodios.

En este trabajo se propone un framework basado en DEVS que provee *building blocks* reutilizables y formalmente especificados para construir entornos de simulación capaces de interactuar con agentes de RL. Mientras que numerosos problemas de control se modelan como Procesos de Decisión de Markov (MDP; Puterman, 1990), los escenarios con eventos estocásticos no controlables y concurrentes requieren Procesos de Decisión Semi-Markovianos Generalizados (GSMDP; Rachelson et al., 2008); el framework los modela como GSMDP dentro de DEVS, definiendo modelos atómicos componibles —*StateObserver*, *EventGenerator* y *Agent*— que encapsulan el protocolo de interacción. Su naturaleza modular y jerárquica permite componerlos en entornos de complejidad variable, manteniendo el foco en el modelado del dominio.

Desarrollo

El entorno de simulación DEVS percibido como un GSMDP

La Figura 1 presenta la arquitectura del framework propuesto. El entorno de simulación DEVS modela el problema de control como un GSMDP: las transiciones de estado son producidas tanto por las acciones del agente (a_t) como por eventos exógenos (e_t), generando observaciones (s_t) y recompensas (r_t). El bloque de *percepción* es el que constituye la vista GSMDP del entorno: intercepta las transiciones que ocurren en el modelo DEVS y las traduce al par observación-recompensa (s_{t+1}, r_{t+1}) que el agente consume en cada paso de decisión.

Esta lectura habilita el aprendizaje por refuerzo sobre un modelo DEVS sin alterar su semántica. En un MDP clásico el tiempo entre decisiones es irrelevante y las transiciones ocurren solo por la acción elegida; en un modelo DEVS realista coexisten múltiples procesos estocásticos concurrentes —cada uno con su distribución de tiempos entre eventos— que compiten por modificar el estado al margen del agente. Esa estructura —tiempo continuo observable, concurrencia de eventos no controlables y transiciones disparadas por la carrera entre relojes— caracteriza a un GSMDP (Rachelson et al., 2008) y DEVS la representa de forma nativa mediante su función de avance de tiempo y su tratamiento de eventos simultáneos. El agente no percibe, entonces, la trayectoria completa del modelo, sino la secuencia de estados en los que se le solicita una decisión, junto con la recompensa acumulada desde su decisión anterior: los instantes de decisión los define la ocurrencia de eventos y no un reloj de muestreo uniforme.

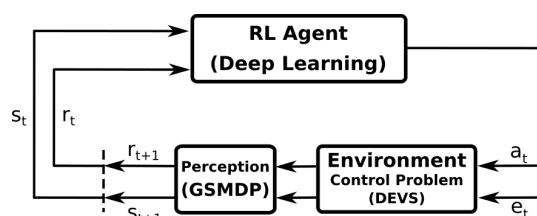


Figura 1. Arquitectura del framework propuesto basado en DEVS para simulación integrada con RL: el agente percibe el entorno de simulación DEVS como un GSMDP

El agente de RL es un componente *pluggable* que interactúa con la simulación a través de puertos DEVS estandarizados: distintos algoritmos pueden conectarse al mismo modelo DEVS y un mismo agente puede interactuar con entornos diferentes.

Modelo formal

A partir de la percepción GSMDP descrita previamente se define el modelo DEVS formal de la Figura 2. El modelo acoplado *SimulationEnvironment* tiene dos componentes principales: el *Agent* y el *GSMDP*. Este último es un DEVS acoplado compuesto por los modelos que representan el problema de control, y expone los puertos “action” (entrada) y

“step” (salida) para la interacción con el agente. Conceptualmente el agente es un bloque intercambiable externo al entorno, pero a los fines de la simulación se acopla *dentro* del *SimulationEnvironment* para ejecutar todo bajo un único simulador DEVS; la separación lógica se preserva porque ambos se comunican solo por los puertos “action” y “step”. En las especificaciones que siguen se omiten, por ser estándar, los conjuntos $X=\{(p,v) \mid p \in InPorts, v \in X_p\}$ y $Y=\{(p,v) \mid p \in OutPorts, v \in Y_p\}$.

Su componente central es el *StateObserver* —Ecuación 1—, un modelo DEVS atómico paralelo reutilizable que mantiene el estado global $s = (o, r, \sigma) \in S$ —observación, última recompensa y avance de tiempo— y lo actualiza ante eventos exógenos o acciones del agente recibidos por el puerto “event”. Se instancia con implementaciones específicas del dominio de las funciones de transición (*tf*), recompensa (*rf*), acciones habilitadas (*enabledA*) y eventos activos (*activeE*). Sus puertos de salida “agent” y “generator” envían el estado al agente y activan o pasivan los *EventGenerator*.

$$\begin{aligned}
 StateObserver &= (X, Y, S, ta, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda) \\
 Event &= \{event_i \mid i = 1, \dots, n\}; \quad A = \text{conjunto de acciones} \\
 InPorts &= \{“event”\}; \quad X_{event} = Event \\
 OutPorts &= \{“generator”, “agent”\}; \quad Y_{generator} = \{true, false\} \\
 Y_{agent} &= \{(o, r, d, A_o) \mid o \in \mathbb{R}^*, r \in \mathbb{R}, \\
 &\quad d \in \{true, false\}, A_o \subseteq A\} \\
 S &= \{(o, r, \sigma) \mid o \in \mathbb{R}^*, r \in \mathbb{R}, \sigma \in \mathbb{R}_0^+\}; \quad ta(s) = \sigma \\
 tf: \mathbb{R}^* \times Event &\rightarrow \mathbb{R}^*; \quad done: \mathbb{R}^* \rightarrow \{true, false\} \\
 rf: \mathbb{R}^* \times Event \times \mathbb{R}^* \times \mathbb{R}_0^+ &\rightarrow \mathbb{R} \\
 enabledA: \mathbb{R}^* &\rightarrow \{a \mid a \subseteq A\}; \quad activeE: \mathbb{R}^* \rightarrow \{true, false\} \\
 \delta_{ext}(o, r, \sigma, e, (p, v)) &= \begin{cases} (tf(o, v), rf(o, v, tf(o, v), e), 0) \\ \text{si } p = “event” \wedge v \in X_p \\ (o, r, \sigma - e) &\text{en otro caso} \end{cases} \\
 \delta_{int}(o, r, \sigma) &= (o, 0, \infty) \\
 \delta_{con}(o, r, \sigma, e, x) &= \delta_{ext}(\delta_{int}(o, r, \sigma), 0, x) \\
 \lambda(o, r, \sigma) &= \begin{cases} (“generator”, activeE(o)) \\ (“agent”, \{o, r, done(o), enabledA(o)\}) \end{cases}
 \end{aligned}$$

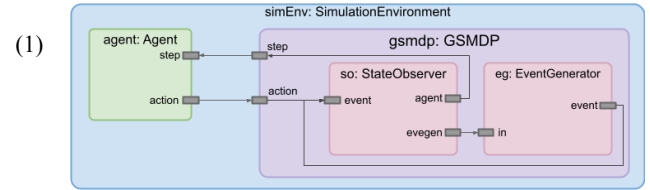


Figura 2. Arquitectura del modelo DEVS acoplado para simulación integrada con RL

El *EventGenerator* —Ecuación 2— genera los eventos exógenos dependientes del tiempo, uno por cada tipo de evento estocástico concurrente. Alterna entre fases pasiva y activa: al ser activado por el puerto “in”, muestrea el tiempo del próximo evento a partir de una distribución de probabilidad y lo emite por el puerto “out”. Aunque la especificación usa una distribución normal a modo ilustrativo, el muestreo está encapsulado en el modelo atómico y puede reemplazarse por cualquier otra distribución. Son estos generadores concurrentes los que materializan la naturaleza GSMDP del entorno. Por su parte, el componente *Agent* —Ecuación 3— es un modelo atómico paralelo intercambiable que envuelve cualquier algoritmo de RL detrás de una interfaz DEVS estandarizada: recibe una tupla *Step* (estado actual *o*, recompensa *r*, indicador de terminación y acciones permitidas *A_o*) por el puerto “step”, la procesa mediante la función de observación *obs* para seleccionar una acción de control y la emite por el puerto “action”.

$$\begin{aligned}
 EventGenerator &= (X, Y, S, ta, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda) \\
 NextEventSigma &\sim \mathcal{N}(eventMean, eventStd) \\
 Event &= \{event_i \mid i = 1, \dots, n\} \\
 EvtActivation &= \text{un conjunto arbitrario} \\
 InPorts &= \{“in”\}; \quad X_{in} = EvtActivation \\
 OutPorts &= \{“out”\}; \quad Y_{out} = Event \\
 S &= \{(phase, ev, \sigma) \mid ev \in Event, \sigma \in \mathbb{R}_0^+, \\
 &\quad phase \in \{“passive”, “active”\}\} \\
 s_0 &= (“passive”, null, \infty); \quad ta(s) = \sigma \\
 \delta_{ext}(phase, ev, \sigma, e, (“in”, v)) &= \begin{cases} (“active”, ev, NextEventSigma) \\ \text{si } phase = “passive” \wedge v = 1 \\ (“active”, ev, \sigma - e) \\ \text{si } phase = “active” \wedge v = 1 \\ (“passive”, ev, \infty) &\text{en otro caso} \end{cases} \\
 \delta_{int}(phase, ev, \sigma) &= \begin{cases} (“active”, ev, NextEventSigma) \\ \text{si } phase = “active” \\ (“passive”, ev, \infty) &\text{en otro caso} \end{cases} \\
 \delta_{con}(phase, ev, \sigma, e, x) &= \delta_{ext}(\delta_{int}(phase, ev, \sigma), 0, x) \\
 \lambda(phase, ev, \sigma) &= (“out”, ev) \quad \text{si } phase = “active”
 \end{aligned}$$

(2)

$$\begin{aligned}
 Agent &= (X, Y, S, ta, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda) \\
 A &= \text{conjunto de acciones del entorno} \\
 Step &= \{(o, r, done, a) \mid o \in \mathbb{R}^*, r \in \mathbb{R}, \\
 &\quad done \in \{true, false\}, a \subseteq A\} \\
 InPorts &= \{“step”\}; \quad X_{step} = Step \\
 OutPorts &= \{“action”\}; \quad Y_{action} = A \\
 S &= \{(ev, \sigma) \mid ev \in Event, \sigma \in \mathbb{R}_0^+\} \\
 s_0 &= (null, \infty); \quad obs: Step \rightarrow A; \quad ta(s) = \sigma \\
 \delta_{ext}(ev, \sigma, e, (p, v)) &= \begin{cases} (obs(v), 0) &\text{si } p = “step” \\ (ev, \infty) &\text{en otro caso} \end{cases} \\
 \delta_{int}(ev, \sigma) &= (null, \infty) \\
 \delta_{con}(ev, \sigma, e, x) &= \delta_{ext}(\delta_{int}(ev, \sigma), 0, x) \\
 \lambda(ev, \sigma) &= (“action”, ev) \quad \text{si } ev \neq null
 \end{aligned}$$

(3)

El framework fue implementado en *Java* sobre el motor de simulación *DEVSJAVA* (<https://sourceforge.net/projects/devs-suitesim/>) y provee como infraestructura lista para usar la clase abstracta *Agent* —con implementaciones preconstruidas de DDQN, A3C y PPO—, *SimulationEnvironment*, *Step* y *WorkerCoordinator*. En consecuencia, el único código específico del dominio requerido es una implementación de la interfaz *EnvironmentLogic* más la configuración de los eventos exógenos del problema. Esa interfaz obliga a proveer, para cada dominio, precisamente las funciones que el modelo formal del *StateObserver* —Ecuación 1— deja sin especificar: transición *tf*, recompensa *rf*, acciones habilitadas *enabledA*, eventos activos *activeE* —que activa o pasiva los *EventGenerator*— y terminación *done*.

Caso de estudio

Para validar la propuesta se utiliza el problema de replanificación presentado en Šemrov et al. (2016), denominado *A simple three-stop railway*, siguiendo la metodología de Beccaria et al. (2021): un agente decide los tiempos de partida para evitar demoras y bloqueos. La topología consta de tres estaciones (A, B y C) unidas por una vía única de cinco secciones de bloqueo con capacidad unitaria; las estaciones admiten hasta tres trenes. Ocurren arribos de trenes no controlables y dos acciones del agente —agregar demora a la partida de un tren y *No Operation* (NOP)—.

La función de recompensa (Ecuación 4) penaliza cada acción de demora según su magnitud *m*, incentivando demoras mínimas. Al finalizar la simulación ($s'_t = 3000$) se aplica una penalización de -1000 por cada tren que no alcanzó su posición de destino *obj_i*, ya sea por bloqueo o por demora excesiva.

$$(4) \quad r(add_i(m), s') = \begin{cases} -m & \text{si } s'_t < 3000 \\ -\sum_{i=1}^3 (s'_{pos_i} \neq obj_i) * 1000 & \text{en otro caso} \end{cases}$$

Los espacios de estados y de eventos del entorno se detallan en las Tablas 1 y 2.

Variable	Descripción
$pos_i \in [0, 15200] \subset \mathbb{R}$	Posición del tren <i>i</i> en metros ($i \in \{1, 2, 3\}$)
$speed_i \in [-27, 27] \subset \mathbb{R}$	Velocidad del tren <i>i</i> en m/s
$trains_j \in \{0, 1, 2, 3\}$	Trenes en la sección de bloqueo <i>j</i> ($j \in \{1, \dots, 8\}$)
$available_j \in \{0, 1\}$	Disponibilidad de la sección de bloqueo <i>j</i>
$t \in [0, 3000] \subset \mathbb{R}^+$	Tiempo de simulación en segundos

Tabla 1. Espacio de estados del entorno ferroviario

Evento	Acción	Descripción
$arrival_i$	No	Se dispara cada vez que un tren en circulación arriba a una estación.
$add(m)$	Sí	<i>m</i> minutos agregados al tiempo de partida del tren siguiente <i>i</i> .
<i>NOP</i>	Sí	Acción <i>No Operation</i> .

Tabla 2. Espacio de eventos del entorno ferroviario

Configuración experimental

Utilizando los building blocks propuestos se compuso el entorno para este problema (Figura 3): se implementó *EnvironmentLogic* para el dominio ferroviario y el entorno se compuso con un *StateObserver* y tres *ExogenousEventGenerator* (uno por arribo de tren), más un generador *FinalEvent* que garantiza la percepción de la recompensa final. Luego se acopló el *Agent*, completando el modelo integrado con RL.

Siguiendo a Šemrov et al. (2016), la tabla horaria define tres trenes con velocidades de 70 y 100 km/h; al inicio de cada episodio un tren aleatorio recibe una demora uniforme de 7 a 10 minutos que, sin intervención, produce un bloqueo. Para evaluar la modularidad del framework se seleccionaron tres implementaciones de agentes preconstruidas, empleando la biblioteca DeepLearning4j: (<https://deeplearning4j.konduit.ai>) *DDQN* (van Hasselt et al., 2016), *A3C* (Mnih et al., 2016) y *PPO* (Schulman et al., 2017). El intercambio entre estas arquitecturas fundamentalmente distintas —basada en valor, actor-crítico y gradiente de política— no requirió ningún cambio en el entorno DEVS: únicamente se sustituyó el componente *Agent*. Cada agente fue entrenado durante 10.000 episodios (3.000 pasos cada uno) en cinco sesiones con distintas semillas aleatorias, con los hiperparámetros de los trabajos originales; la implementación (<https://github.com/ezequielbeccaria/RLDEVS4j-SingleTrackRailway>) se ejecutó en un AMD Ryzen 7 1800 (8 núcleos, 32 GB de RAM).

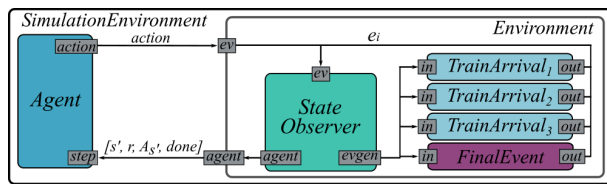


Figura 3. Entorno de simulación DEVS compuesto a partir de los building blocks propuestos, con sus componentes y acoplamientos

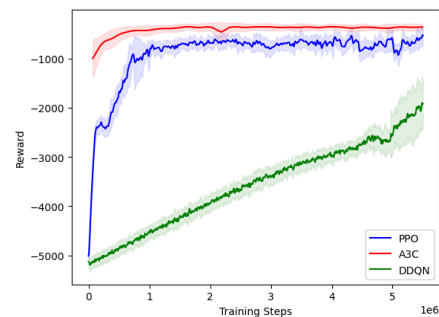


Figura 4. Curvas de recompensa de entrenamiento (media $\pm$ desvío estándar sobre cinco semillas aleatorias) para DDQN, A3C y PPO

Para validar el entorno se compararon sus trazas contra una implementación equivalente en Gymnasium (Towers et al., 2024) mediante *Dynamic Time Warping* (DTW; Bringmann et al., 2024) en cuatro escenarios, obteniéndose una distancia media de $\mu = 0,31$ ($\sigma = 1,51$). El elevado desvío refleja la distinta densidad de eventos entre escenarios; aun así, todas las distancias individuales se mantuvieron por debajo de umbrales consistentes con un comportamiento cualitativo equivalente.

Resultados

Se comparó el desempeño de la política aprendida por cada agente (A3C, DDQN y PPO) en doce escenarios con 7 a 10 minutos de demora por tren (Tabla 3).

	7 min			8 min			9 min			10 min			Resumen	
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3	Media	Desv.
DDQN	-420	-240	-420	-480	-240	-3360	-600	-360	-3420	-3240	-840	-360	-1165,0	1265,57
A3C	-120	-360	-480	-120	-360	-480	-120	-360	-480	-120	-360	-480	-320,0	149,66
PPO	-0	-3360	-300	-0	-360	-300	-60	-360	-300	-120	-360	-3360	-740,0	1179,15

Tabla 3. Resultados de los doce escenarios de prueba (7 a 10 minutos de demora inicial sobre los trenes T1, T2 y T3); en negrita, el mejor desempeño de cada escenario

Los resultados evidencian la ventaja principal del framework: el mismo entorno DEVS permitió evaluar tres arquitecturas de RL fundamentalmente distintas sin modificación alguna del modelo. Las curvas de aprendizaje (Figura 4) explican las diferencias: A3C converge más rápido y es el más estable, consistente con su menor varianza en prueba; PPO alcanza una recompensa cercana a la óptima con una trayectoria más ruidosa; y DDQN sigue mejorando al agotarse el presupuesto de entrenamiento, lo que explica su mayor varianza. Las diferencias reflejan, entonces, la velocidad de convergencia bajo un presupuesto fijo de 10.000 episodios más que la calidad asintótica de la política.

Conclusiones

El framework propuesto provee building blocks DEVS reutilizables y formalmente especificados —*StateObserver*, *EventGenerator* y *Agent*— que encapsulan el protocolo de interacción con el agente, soportan despliegues de un único trabajador o paralelos, y habilitan la evaluación de múltiples algoritmos de RL sobre el mismo entorno sin modificaciones, como se validó sobre el problema *A simple three-stop railway*. Al eliminar la necesidad de implementar desde cero la infraestructura de integración, el enfoque desplaza el foco hacia el modelado de los procesos estocásticos concurrentes y las funciones de recompensa del dominio. Además, los building blocks pueden incorporarse a modelos DEVS preexistentes, integrando agentes de RL en entornos ya validados sin rediseñarlos.

El trabajo futuro incluye incorporar verificación formal para validar la corrección de los entornos construidos, explorar IA generativa para generar nuevas configuraciones de entorno, desarrollar una capa puente en Python que mejore la interoperabilidad con el ecosistema de RL dominante, y evaluar el framework en escenarios más complejos —mayores espacios de estados, más concurrencia de eventos o coordinación multiagente—.

Utilización de IA generativa / herramientas IA

Declaro la utilización de la herramienta de IA Claude (Anthropic) para la adaptación de formato y la traducción al español de un trabajo previo de propia autoría. El contenido científico, el modelo formal, la implementación y los resultados

experimentales son de autoría propia. Confirmando que luego de la utilización, los contenidos obtenidos fueron revisados, constatados y editados, asumiendo plena responsabilidad por el contenido presentado en el presente trabajo.

Referencias

- Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6), 26–38. <https://doi.org/10.1109/MSP.2017.2743240>
- Beccaria, E., Bogado, V., & Palombarini, J. A. (2021). A DEVS based methodological framework for reinforcement learning agent training. *IEEE Latin America Transactions*, 19(4), 679–687. <https://doi.org/10.1109/TLA.2021.9448551>
- Blas, M. J., Leone, H., & Gonnet, S. (2022). DEVS-based formalism for the modeling of routing processes. *Software and Systems Modeling*, 21(3), 1179–1208. <https://doi.org/10.1007/s10270-021-00928-4>
- Bogado, V., Gonnet, S., & Leone, H. (2017). DEVS-based methodological framework for multi-quality attribute evaluation using software architectures. En *2017 XLIII Latin American Computer Conference (CLEI)* (pp. 1–10). IEEE. <https://doi.org/10.1109/CLEI.2017.8226435>
- Bringmann, K., Fischer, N., van der Hoog, I., Kipouridis, E., Kociumaka, T., & Rotenberg, E. (2024). Dynamic dynamic time warping. En *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (pp. 208–242). Society for Industrial and Applied Mathematics.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of Machine Learning Research*, 48, 1928–1937.
- Puterman, M. L. (1990). Markov decision processes. En D. P. Heyman & M. J. Sobel (Eds.), *Handbooks in operations research and management science* (Vol. 2, pp. 331–434). Elsevier. [https://doi.org/10.1016/S0927-0507\(05\)80172-0](https://doi.org/10.1016/S0927-0507(05)80172-0)
- Rachelson, E., Quesnel, G., Garcia, F., & Fabiani, P. (2008). A simulation-based approach for solving generalized semi-Markov decision processes. En *Proceedings of the 18th European Conference on Artificial Intelligence* (pp. 583–587). IOS Press. <https://doi.org/10.3233/978-1-58603-891-5-583>
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms* (arXiv:1707.06347). arXiv. <https://doi.org/10.48550/arXiv.1707.06347>
- Šemrov, D., Marsetič, R., Žura, M., Todorovski, L., & Srdic, A. (2016). Reinforcement learning approach for train rescheduling on a single-track railway. *Transportation Research Part B: Methodological*, 86, 250–267. <https://doi.org/10.1016/j.trb.2016.01.004>
- Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., & Younis, O. G. (2024). *Gymnasium: A standard interface for reinforcement learning environments* (arXiv:2407.17032). arXiv.
- van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2094–2100. <https://doi.org/10.1609/aaai.v30i1.10295>
- Zeigler, B. P., Muzy, A., & Kofman, E. (2019). Introduction to systems modeling concepts. En B. P. Zeigler, A. Muzy & E. Kofman (Eds.), *Theory of modeling and simulation* (3.^a ed., pp. 3–25). Academic Press. <https://doi.org/10.1016/B978-0-12-813370-5.00009-2>