Framework de Simulación Interactiva e Inmersiva para Análisis Dinámico de Datos

Gisela Confalonieri, Ezequiel Pecker-Marcosig, Julián Ottomano, and Rodrigo Castro

Departamento de Computación (FCEyN, UBA) e ICC (UBA/CONICET), Ciudad Universitaria, Pabellón 0+Infinito, C1428EGA, Buenos Aires, ARGENTINA
gconfalonieri@dc.uba.ar, emarcosig@dc.uba.ar, jottomano@dc.uba.ar, rcastro@dc.uba.ar

Abstract. El manejo de visualizaciones efectivas para el análisis de datos es fundamental para facilitar la toma de decisiones informadas en múltiples dominios. Esto incluye datos tomados tanto de escenarios reales así como virtuales o simulados, que permitan analizar escenarios hipotéticos. La posibilidad de interactuar en tiempo real con datos de simulaciones, reflejados en visualizaciones inmersivas, puede facilitar la comprensión de los sistemas bajo estudio. Este trabajo propone un enfoque innovador para la exploración visual, inmersiva e interactiva de datos y simulaciones, destacando la importancia de cómo se muestran e interactúa con los datos, el modelo de simulación y el entorno físico, bajo el concepto de simulación interactiva. Para ello, presentamos un framework que combina un simulador de sistemas híbridos con un motor de videojuegos, permitiendo la interacción en tiempo real con simulaciones a través de visualizaciones realistas. Mostramos la utilidad de este enfoque en la interacción humano-computadora mediante tres casos de estudio, sustentados por una sala de visualización inmersiva.

Keywords: Simulación interactiva, Datos, Visualización inmersiva

1 Introducción y descripción del problema

Los procesos de toma de decisiones complejas requieren, frecuentemente, de la participación de una variedad de expertos en diferentes dominios del conocimiento, llevando a cabo un análsis colaborativo de datos. Para ayudar a las partes interesadas a tomar decisiones informadas, resulta fundamental contar con herramientas que den soporte al razonamiento. La visualización de datos es esencial para este propósito.

Existen diversos campos de estudio con foco en la visualización de datos y en la interacción humana para su análisis, como *Visual Analytics* (Cui, 2019) e *Immersive Analytics* (Chandler et al., 2015). La interacción con los datos y su visualización son valiosas para una exploración de datos flexible que ayude a revelar información novedosa (Matković et al., 2018).

Vale la pena señalar que los datos a ser analizados no siempre provienen del mundo real, por ejemplo, cuando hay una necesidad de explorar escenarios

futuros, o cuando es peligroso, difícil o incluso imposible de obtener datos del mundo real. En tales casos, los datos del mundo real pueden enriquecerse con datos virtuales generados de modelos de sistemas, que a su vez son necesarios para evaluar cómo las acciones de los usuarios impactarían el sistema.

Los modelos de interés por lo general son complejos, abarcando múltiples formalismos matemáticos y dinámicas híbridas. Por ello, a menudo carecen de soluciones analíticas y deben recurrir a simulaciones.

Con el propósito de promover una experiencia inmersiva, necesitamos considerar las formas en que se muestran y en que se interactúa con los datos, los modelos de simulación y el entorno. En contraste con *ensemble analysis* (Matković et al., 2018), donde los parámetros de modelos están fijos para cada corrida de simulación, introducimos el concepto de *simulación interactiva*, en la que un usuario pueda cambiar sus valores en tiempo real a medida que evoluciona la simulación, conduciendo a resultados que no podrían ser capturados por abordajes previos, pero manteniendo el concepto de *exploración del espacio de parámetros* para encontrar conjuntos de valores que producen respuestas deseadas.

Desde el punto de vista de usuario, la idea de usar el espacio a su alrededor como espacio de trabajo a través de *interfaces de usuario naturales* (*natural user interfaces*, NUI) resulta en una exploración de datos más intuitiva. Esto puede lograrse mediante el uso de dispositivos de entrada portátiles y/o inalámbricos, y dispositivos de visualización varios, como entornos virtuales automáticos (CAVE) para grupos o dispositivos montados en la cabeza (HMD) para individuos. Todos ellos se enmarcan dentro del campo de *interacción humano-máquina* (*human-computer interaction*, HCI), representado en la Fig. 1. Desde

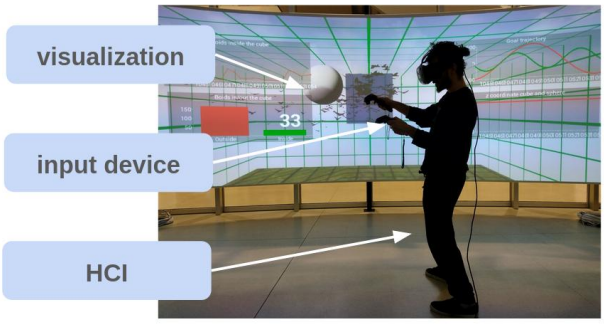


Fig. 1: Usuario interactuando con simulación en un entorno inmersivo.

la perspectiva del simulador, las acciones llevadas a cabo por un usuario, como modificar el valor de un parámetro, pueden interpretarse como eventos discretos. De este modo, la evolución del sistema se ve interrumpida por eventos de entrada que llevan información que afecta el comportamiento del modelo a partir de ese momento. El simulador debe ser capaz de manejar eventos discretos

eficientemente mientras resuelve numéricamente ecuaciones diferenciales de las partes continuas, y refleja los efectos de las acciones *más rápido que tiempo real* (*fastar-than-real-time*, FTRT).

Estos puntos de vista confluyen en un framework en el que integramos el simulador de alta precisión PowerDEVS (Bergero and Kofman, 2011) con el motor de juegos Unreal Engine (Epic Games, 2025) para crear representaciones realistas de *sistemas socio-ciber-físicos* (SCPS) simulados, utilizando visualizaciones flexibles y atractivas de datos reales y virtuales combinados.

La estructura de este trabajo es la siguiente. La Sección 2 introduce conceptos y herramientas que serán utilizados a lo largo del trabajo, mientras que la Sección 3 presenta el framework propuesto. La Sección 4 describe tres casos de estudio que resaltan distintos modos de utilización de este framework, destacando la capacidad de intervenir simulaciones y los datos generados de manera interactiva en tiempo de ejecución. Finalmente, la Sección 5 concluye el trabajo.

2 Conceptos Preliminares

2.1 El Formalismo de Especificación de Sistemas de Eventos Discretos (DEVS)

Para el modelado y simulación de sistemas dinámicos híbridos, utilizamos el formalismo de *especificación de sistemas de eventos discretos* (DEVS) (Zeigler et al., 2018). DEVS puede representar cualquier sistema discreto (de eventos discretos o de tiempo discreto) siempre y cuando produzca un número finito de eventos en un intervalo de tiempo finito. DEVS también permite aproximar sistemas continuos con cualquier nivel de precisión requerido, recurriendo a los métodos numéricos de *sistemas de estados cuantizados* (QSS) (Castro et al., 2024), y expresar el comportamiento estocástico para sistemas híbridos generalizados (Castro et al., 2010).

Un aspecto destacable de DEVS es su estructura modular y jerárquica, basada en modelos estructurales (acoplados) y de comportamiento (atómicos), lo que promueve la construcción incremental de modelos complejos, la reutilización y sustitución robustas de módulos, y la construcción jerárquica de modelos. Esto fomenta la definición de bibliotecas de modelos.

Formalmente, un modelo *Atómico* DEVS es un bloque de construcción mínimal definido como $M_A = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$ donde X es el conjunto de eventos de entrada, Y es el conjunto de eventos de salida, y S es el conjunto de estados. Cuatro funciones dinámicas definen el comportamiento del modelo: δ_{int} (función de transición interna, para comportamiento autónomo), δ_{ext} (función de transición externa, para recepción de eventos y comportamiento reactivo), λ (función de salida, para emisión de eventos) and ta (función de avance de tiempo, para sincronización autónoma). Cuando el modelo atómico alcanza un estado $s \in S$, y en ausencia de eventos de entrada externos, permanecerá en s durante un tiempo $ta(s)$.

Un modelo *Acoplado* DEVS es una red de modelos atómicos y/o acoplados definida como $M_C = \langle X_{self}, Y_{self}, D, \{M_d\}, \{I_d\}, \{Z_{d,j}\}, Select \rangle$ donde $self$

es el modelo acoplado en sí, X_{self} e Y_{self} son los conjuntos de valores de entrada y salida (respectivamente) de la estructura, D es el diccionario de componentes conectados que pertenecen a $self$, y M_d ($d \in D$) es cualquier otro mdelo, acoplado o atómico. Para cada $d \in D \cup \{self\}$, I_d es el conjunto de modelos que influencian (*influyentes*) al subsistema d. Para cada $j \in I_d$, $Z_{d,j} : Y_d \to X_j$ es la función de traducción de d a j, mientras que $Select : 2^D \to D$ es una función de desempate para eventos simultáneos.

DEVS presenta un algoritmo simulador abstracto generalizado, con dos clases principales de objetos: *simuladores* y *coordinadores*. La ejecución de cada modelo atómico está controlada por su *simulador*, mientras que los *coordinadores* gestionan la coordinación de los modelos acoplados y se encargan de sincronizar sus simuladores y coordinadores secundarios. En la cima de esta jerarquía, un *Coordinador raíz* se encarga de activar nuevos ciclos de simulación y gestionar el avance del tiempo de simulación global.

En el contexto de análisis de escenarios en entornos de visualización inmersiva, la interacción frecuente con un usuario (como la modificación interactiva de parámetros) y/o la ejecución de acciones cuando alguna variable alcanza una meta/límite, constituyen eventos discretos que generan discontinuidades en la evolución de las variables continuas, pero cuyo tratamiento con DEVS y los métodos numéricos QSS es trivial (se evitan algoritmos iterativos de resincronización entre las partes discreta y continua). Además, la actividad causada por esos eventos se resuelve localmente (en los modelos donde ocurren cambios relevantes). Todo esto resulta en simulaciones interactivas más eficientes.

2.2 La herramienta de simulación PowerDEVS

El kit de herramientas PowerDEVS (Bergero and Kofman, 2011) es particularmente adecuado para la simulación eficiente de sistemas dinámicos híbridos. Esto se debe en parte a que es el simulador DEVS insignia para la familia de métodos numéricos QSS (Castro et al., 2024). Una ventaja distintiva de PowerDEVS en comparación con otros simuladores DEVS es su rendimiento (lo que lo hace recomendable para aplicaciones en donde el desempeño de la simulación es un aspecto clave) y ha sido recomendado para modeladores no familiarizados con DEVS y para usuarios que desean usar DEVS para la simulación de sistemas dinámicos híbridos (Van Tendeloo and Vangheluwe, 2017).

La versión básica de PowerDEVS comprende cuatro programas independientes. El *Editor de Modelos* es la interfaz gráfica de usuario (GUI) que proporciona un lienzo para construir y configurar el modelo de simulación. El *Editor de Atómicos* está destinado a describir el comportamiento de los modelos atómicos. El *Pre-Procesador* genera código C++ a partir de la representación del modelo gráfico, que luego se compila para producir un ejecutable independiente que incluye la especificación del modelo y el motor de simulación. Por último, la *Interfaz de Simulación* es una interfaz gráfica para el manejo de la simulación que permite la ejecución interactiva de simulaciones. En la *Vista de Diseño* (ver Fig. 2), se construye un modelo interconectando representaciones gráficas de modelos atómicos y acoplados de DEVS (bloques con conexiones de entrada/salida

que se pueden arrastrar, soltar e interconectar), incluida la configuración de sus parámetros de modelo. La *Vista de Simulación*, proporcionada por la *Interfaz de Simulación* o a través de la línea de comandos con el ejecutable producido por el *Pre-Procesador*, permite establecer los parámetros de simulación (por ejemplo, el tiempo final de simulación, el número de ejecuciones, etc.) y verificar la evolución del tiempo de simulación. Más allá de los eventos discretos genera-

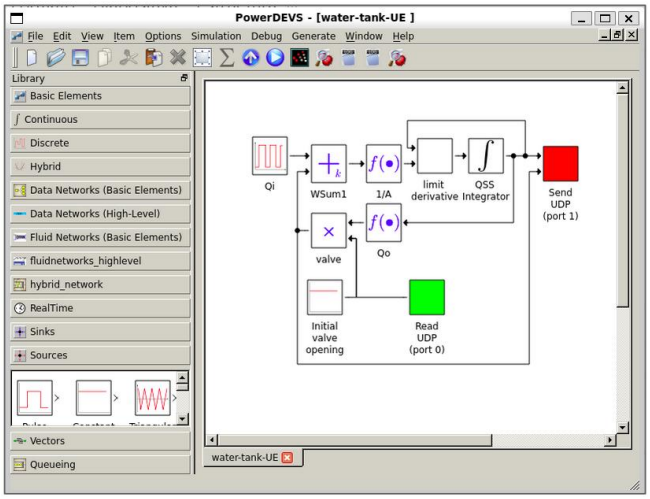


Fig. 2: Captura del Editor de Modelos (GUI) de PowerDEVS. El lienzo central muestra un modelo compuesto por atómicos interconectados con flechas que indican el sentido de comunicación. La barra de la izquierda muestra las bibliotecas de modelos disponibles.

dos internamente por un modelo de PowerDEVS, el simulador abstracto DEVS fue adaptado para recibir eventos externos desde fuera de PowerDEVS (Pecker-Marcosig et al., 2018). Cada vez que un modelo atómico requiere escuchar un puerto de comunicación dado, se lanza un *listener* en un hilo de ejecución secundario, que continuamente monitorea ese puerto. Los eventos entrantes que llegan en cualquier momento se añaden en una cola de mensajes común a todos los *listeners*. Durante el tiempo de inactividad, el *Coordinador raíz* revisa la cola de mensajes y procesa los mensajes entrantes, mapeándolos como eventos externos para los atómicos que estaban "escuchando". Un mecanismo similar se encuentra disponible para que un modelo atómico envíe eventos afuera de PowerDEVS. Al momento, este esquema permite lanzar *listeners* para escuchar en sockets UDP.

2.3 Motor de Juegos Unreal Engine

Unreal Engine (UE) es un motor de gráficos 3D para videojuegos desarrollado por la compañía Epic Games (Epic Games, 2025). Diseñado originalmente para

ser la tecnología subyacente de los videojuegos, y siendo utilizado en varios títulos de alto perfil con gran capacidad gráfica, también ha sido adoptado por otras industrias, especialmente el cine y la televisión. Por ejemplo, se ha utilizado en la industria cinematográfica para crear escenarios virtuales que pueden seguir el movimiento de la cámara alrededor de actores y objetos, y renderizarse en tiempo real en grandes pantallas LED y sistemas de iluminación ambiental.

No obstante, Unreal Engine también está ganando terreno en ámbitos no creativos debido a su disponibilidad y funcionalidades. Los desarrollos con apariencia de videojuegos pero pensados para ámbitos tales como educación y exploración científica, denominados *juegos serios*, buscan aprovechar los mismos recursos que los juegos para entretenimiento, incluyendo hardware específico y entornos inmersivos. Se ha utilizado, por ejemplo, como base para herramientas de realidad virtual para exploración de moléculas, y herramientas de simulación multijugador para entrenamiento de personal, entre tantas otras.

Unreal Engine está escrito en lenguaje C++ y ofrece un alto grado de portabilidad, siendo compatible con una amplia gama de plataformas de escritorio, móviles, consolas y realidad virtual. Asimismo, provee el sistema de scripting visual llamado *Blueprint* (Fig. 6c) basado en nodos interconectados, que permite crear y manipular objetos del juego, así como programar su lógica con un enfoque orientado a objertos. Unreal Engine es de código abierto y permite extender la funcionalidad de un proyecto y del editor en sí mediante plugins.

3 Framework de Integración entre PowerDEVS y Unreal Engine

Esta sección presenta el framework desarrollado para la integración entre PowerDEVS (PD) y Unreal Engine (UE). Para establecer la comunicación entre un modelo de PowerDEVS y una aplicación desarrollada con Unreal Engine, tomamos como base mecanismos disponibles desarrollados previamente: (a) *plugins* de comunicación UDP en Unreal Engine, y (b) interfaz de comunicación UDP en PowerDEVS, descripta en la Sección 2.2.

La elección de UDP por sobre TCP nace por la sobrecarga de este último debido a su naturaleza orientada a conexión y su mecanismo de control de congestión, aun reconociendo la incapacidad de UDP para recuperarse de la pérdida de información a nivel de transporte y delegando esta tarea al nivel de aplicación.

Teniendo la posibilidad de comunicar ambos entornos por red en tiempo de ejecución, considerando que UE permite crear aplicaciones gráficas interactivas y ofrece la posibilidad de manejar una diversidad de dispositivos de entrada, y que PD permite trabajar con eventos externos al modelo en tiempo de simulación, logramos cerrar el circuito mediante el cual un usuario puede modificar parámetros del modelo de simulación en PD indirectamente a través de una interfaz de usuario visual diseñada en UE. La Fig. 3 sintetiza esta comunicación y sus efectos.

PD envía periódicamente datos a UE para actualizar una animación representativa. Al mismo tiempo, UE permite la intervención del usuario, como

modificar la posición de un objeto, y envía estos datos de vuelta a PD para ajustar los parámetros del modelo en tiempo real. Este esquema es la base para diversos casos de uso. Aunque el protocolo UDP no garantiza la entrega de todos los datos, esta limitación no representa un problema significativo: una pérdida ocasional de paquete puede generar un retraso imperceptible en UE, similar a un "lag" en una transmisión de video, sin afectar el desarrollo continuo de la simulación. Las acciones del usuario son enviadas como una secuencia de eventos que, ante la eventualidad de una pérdida de un paquete UDP, no alteran sustancialmente la evolución del sistema, ya que esta se basa en secuencias continuas de interacción.

A continuación, la Sección 3.1 explica el uso del plugin utilizado en Unreal Engine para la comunicación UDP, mientras que la Sección 3.2 detalla la implementación llevada a cabo en PowerDEVS a estos efectos.

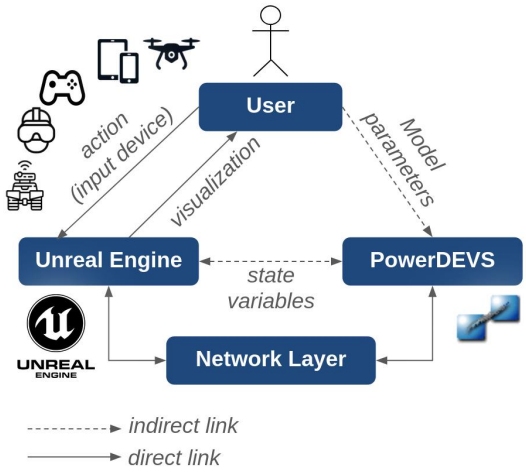


Fig. 3: Integración entre PowerDEVS y Unreal Engine mediante una red de comunicaciones bidireccional que permite la visualización de los resultados de simulación y la modificación de parámetros del modelo de forma concurrente en tiempo de simulación.

3.1 Comunicación UDP en Unreal Engine

Para establecer una comunicación entre una aplicación desarrollada con UE y una aplicación externa (en nuestro caso, PD) se utilizará el plugin *UDP-Communication* (TalTech, 2023). Este plugin fue desarrollado originalmente con el propósito de comunicar un *gemelo digital* desarrollado en UE con modelos en MATLAB/Simulink, y utiliza una trama de datos compuesta por 23 campos (20 campos de tipo `float` seguidos por 3 campos de tipo `unit8`). Los datos a transmitir deben seguir el orden establecido en esta estructura, aunque no es necesario

utilizar todos los campos. El plugin *UDP-Communication* es de código abierto, permitiendo la edición del código fuente para modificar la trama de datos a enviar y recibir.

Este plugin expone su funcionalidad a bloques de Blueprint, permitiendo la recepción y envío de datos por UDP (bloques `Get Data` y `UDPSend Array`), así como la parametrización de la trama a enviar y la desagregación de la trama recibida (bloques `Make UDPData` y `Break UDPData`). En la Figura 6c se presenta un ejemplo de uso de Blueprint, donde las áreas coloreadas en celeste delimitan la lógica de la comunicación: arriba se muestra el mecanismo de recepción de mensajes para luego utilizar los datos recibidos en el comportamiento de otro actor de la escena, y abajo se muestra el mecanismo de creación y envío de mensajes al ocurrir ciertos eventos en la escena.

3.2 Comunicación vía UDP en PowerDEVS

La herramienta de simulación PowerDEVS tiene la capacidad de comunicarse con software o hardware externo mediante sockets de red UDP, utilizando un equema asíncrono de *request-reply full-duplex* (Pecker-Marcosig et al., 2018). En el mecanismo implementado, los modelos atómicos DEVS pueden enviar y recibir mensajes UDP, representándolos como eventos externos en un entorno DEVS (ver Sección 2.2). Tomamos esto como base para el desarrollo de nuevos atómicos que permitan la comunicación con UE. Estos nuevos modelos atómicos fueron recientemente incorporados como parte de la biblioteca `udpcomm` de PowerDEVS.

Estos modelos componen los mensajes de manera dinámica a partir de una cantidad de datos variable y configurable, lo que permite su adaptación a distintos contextos que requieren comunicar diferentes cantidades de datos. El modelo atómico `SendUDP` para el envío de mensajes UDP (bloque rojo en la Figura 2) permite parametrizar la cantidad de datos que quieren enviarse, la dirección IP y el puerto UDP del destino, y el tiempo entre envíos consecutivos. Por su parte, el modelo atómico `ReadUDP` para la recepción de mensajes UDP (bloque verde en la Figura 2) permite parametrizar el puerto UDP de escucha y la cantidad de valores a recibir en un mensaje.

De esta manera, aprovechando la construcción de modelos DEVS como la composición jerárquica de modelos atómicos simples, cada uno de estos modelos atómicos puede conectarse a un modelo más grande, permitiendo enviar una cantidad arbitraria de datos proveniente de una simulación, así como recibir una cantidad arbitraria de datos externos al modelo y utilizarlos para modificar el mismo en tiempo de simulación.

4 Casos de Estudio

En esta sección se presentan tres casos de estudio con un nivel de complejidad incremental utilizando el framework propuesto en la Sección 3.

La Sección 4.1 resalta la capacidad de generar una visualización realista de los resultados de una simulación en tiempo de ejecución. La Sección 4.2 muestra la posibilidad de que un usuario interactúe con la visualización y el modelo,

alterando parámetros de la simulación en tiempo real, y observando inmediatamente sus efectos. La Sección 4.3 añade a lo anterior la capacidad de co-simular escenarios entre múltiples motores, en este caso Unreal Engine y PowerDEVS, donde los resultados en uno impactan en el comportamiento del otro y viceversa.

Todos estos casos fueron también puestos a prueba en una sala de visualización con una pantalla curva de gran tamaño que envuelve al usuario, y con la integración de dispositivos de entrada *hand-held* o inalámbricos, favoreciendo una experiencia inmersiva.

Todos los ejemplos están disponibles públicamente en un repositorio de git (SEDLab, 2023).

4.1 Aeronave

En este primer caso práctico, analizamos la posibilidad de visualizar de manera realista y en tiempo real el resultado de una simulación, aún sin integrar la interacción con un usuario.

Para este escenario, se simula con PowerDEVS un modelo de un vehículo aéreo no tripulado (UAV) de ala fija equipado con un controlador híbrido jerárquico asignado a una misión de vigilancia (Pecker-Marcosig et al., 2023). Los datos de posición y orientación resultantes de la aeronave simulada se envían a Unreal Engine en tiempo real para producir una visualización realista y atractiva en un mapa georreferenciado utilizando el plugin *Cesium for Unreal* (Cesium, 2024).

La Fig. 4 muestra la visualización resultante del UAV en el mapa proyectado en una pantalla de gran tamaño (izquierda), junto con algunos gráficos producidos en PowerDEVS y la arquitectura del modelo (derecha).

Por su parte, la Fig. 5 muestra una captura del modelo de PowerDEVS utilizado. En el acoplado de más bajo nivel, que incluye los lazos de control realimentado de bajo nivel del UAV y su dinámica, se observa que el atómico `sendUDP` conforma una trama de datos con la posición actual del UAV (`<pos_x>`, `<pos_y>`, `<pos_z>`) y su orientación (`<pitch>`, `<roll>`, `<yaw>`) para enviar periódicamente (cada 1 s) a Unreal Engine con el siguiente formato:

$$\texttt{dataUAV} = \langle \texttt{<pos\_x>}, \texttt{<pos\_y>}, \texttt{<pos\_z>}, \texttt{<pitch>}, \texttt{<roll>}, \texttt{<yaw>} \rangle,$$

donde cada elemento de la trama es de tipo `double`.

4.2 Tanque de Agua

Para evaluar la interactividad entre un usuario, una visualización realista y el simulador, se construyó un modelo del proceso de llenado de un tanque de agua. Este modelo se basa en una ecuación diferencial ordinaria (ODE) no lineal que describe el comportamiento de un tanque de agua que recibe un caudal de entrada constante Q_i. El sistema incluye una válvula que regula el caudal de salida Q_o, lo cual influye directamente en el nivel de agua h. La apertura de la válvula puede ser controlada mediante una señal de entrada externa u. La Fig. 6b muestra la representación gráfica del modelo en PowerDEVS.

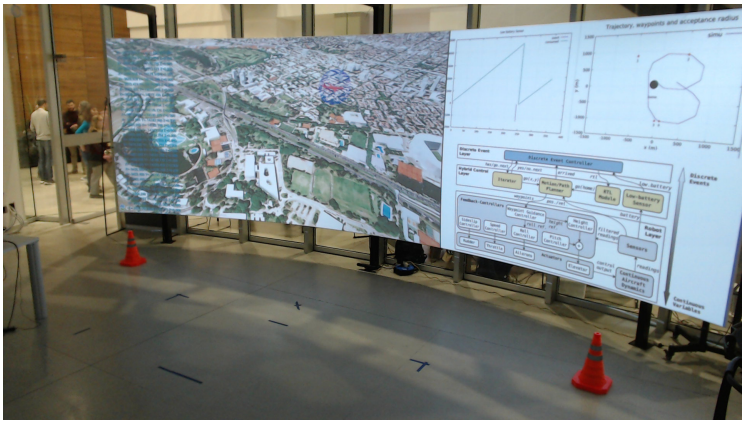


Fig. 4: Visualización en tiempo real de los resultados de simulación del UAV sobre un mapa renderizado 3D geoespacial basado en GIS.

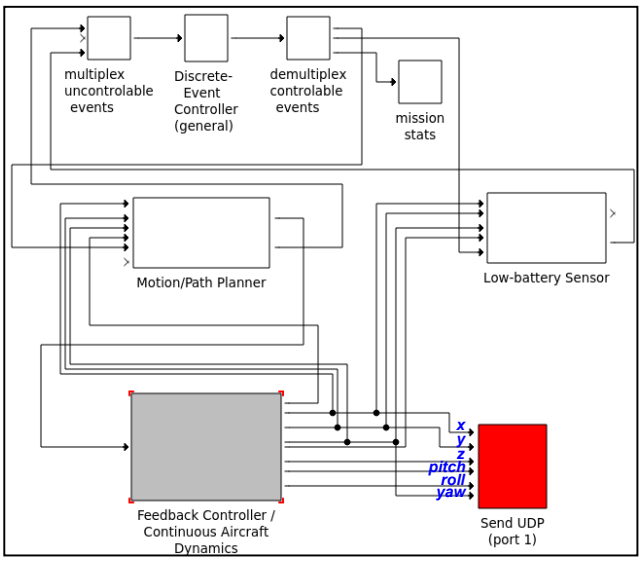


Fig. 5: Captura del modelo en la GUI de PowerDEVS. Arquitectura completa del modelo jerárquico desde el control de la misión hasta el modelo del UAV.

En esta aplicación, un usuario puede influir en el modelo en PowerDEVS mediante la apertura o cierre gradual de una representación gráfica de una válvula en Unreal Engine utilizando un dispositivo de entrada (e.g. *joysick* o *mouse*). En UE también se diseñó una visualización realista del agua dentro del tanque. La Fig. 6a muestra el entorno de la aplicación en Unreal Engine.

En esencia, PD ejecuta el modelo del tanque de agua y transmite periódicamente los valores de nivel de agua h y caudal de salida Q_o a UE mediante sockets UDP. Para esto utiliza un atómico `SendUDP` (Figura 6b) que recolecta los valores de h y Q_0, y periódicamente confecciona y envía una trama de datos con el siguiente formato:

$$\texttt{dataTanque} = \langle \texttt{<nivel>}, \texttt{<caudal\_salida>} \rangle.$$

La aplicación en UE recibe estos mensajes UDP (Figura 6c, arriba) y actualiza la visualización del agua dentro del tanque (Figura 6a, centro) y el gráfico con la evolución temporal del nivel de agua y el caudal de salida (Figura 6a, derecha).

En este caso, la interfaz de usuario de la aplicación en UE permite al usuario interactuar con la representación gráfica de la válvula u (Figura 6a, izquierda), regulando el caudal de salida durante la simulación. En concreto, la aplicación en UE captura los eventos de entrada en la válvula y los comunica al modelo de PD mediante sockets UDP (Figura 6c, abajo). Finalmente, el modelo de PD recibe esos eventos externos a PowerDEVS mediante un atómico `ReadUDP` (Figura 6b), lee el valor de la apertura de la válvula contenido en el valor del evento:

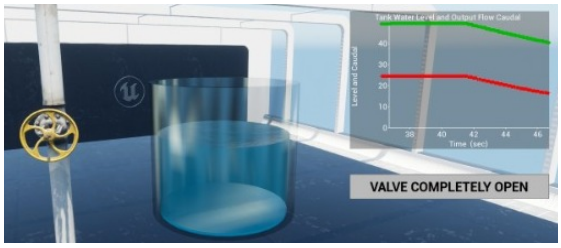
$$\texttt{dataValvula} = \langle \texttt{<apertura\_valvula>} \rangle,$$

en donde el campo `apertura_valvula` es de tipo `double` y toma un valor entre 0 (válvula totalmente cerrada) y 1 (válvula totalmente abierta). Este valor se utiliza para actualziar el valor de u en el modelo de PowerDEVS en tiempo de ejecución.

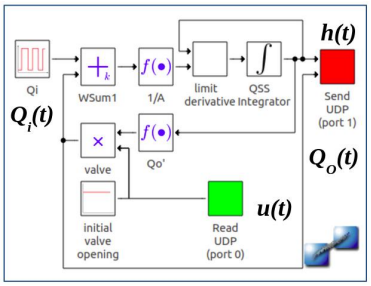
4.3 Bandada de Aves

A diferencia del ejemplo anterior, que incluía un único modelo dinámico ejecutándose en PowerDEVS, el ejemplo de bandada de pájaros está diseñado para explorar la interacción bidireccional entre dos modelos que se ejecutan simultáneamente en PowerDEVS y Unreal Engine (también conocido como *co-simulación*). La Figura 7a muestra la animación en UE, y la Figura 7b muestra la representación gráfica del modelo en PowerDEVS.

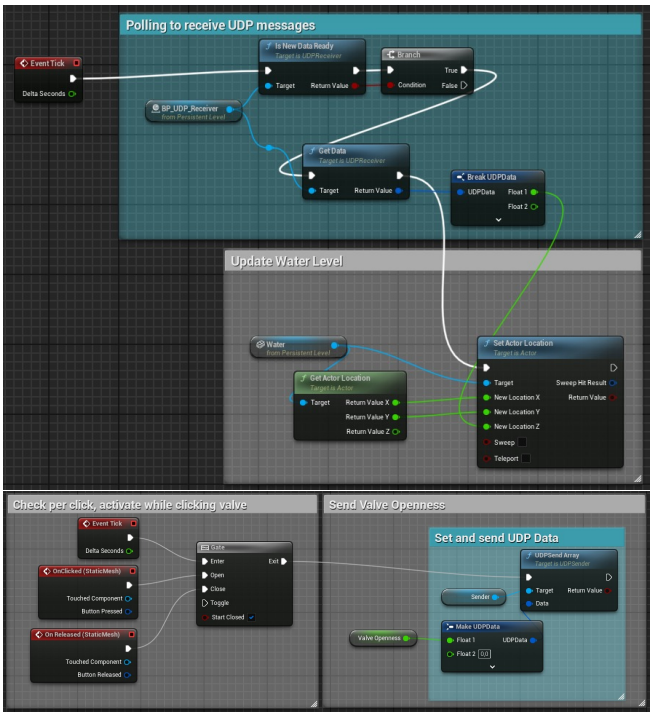
En esencia, integramos un modelo basado en agentes de una bandada compuesta por N aves, desarrollado en UE (utilizando el plugin *Flocks*) y configurado para seguir un objetivo (una esfera) que sigue una trayectoria en forma de 8 generada con PD. UE se encarga de simular el modelo basado en agentes de la bandada y de producir visualizaciones realistas de la bandada, la esfera y un volumen de control que el usuario puede manipular para maximizar el número de aves que cruzan este volumen en un instante dado. Por otra parte, PD ejecuta un modelo adaptativo de la trayectoria en forma de 8.



(a) Captura de la escena en Unreal Engine.



(b) Captura de PowerDEVS.



(c) Capturas de Blueprints en Unreal Engine.

Fig. 6: Manejo interactivo del llenado de un tanque de agua. (a) Visualización interactiva, realista y dinámica del taque y el líquido en UE, (b) Modelo ODE para el comportamiento del tanque de agua, con una válvula para regular el nivel y el flujo de salida, en PD. (c) Uso de Blueprint en UE para manejo de la comunicación UDP.

La co-simulación funciona de la siguiente manera. PowerDEVS simula la trayectoria en forma de 8 de la esfera y envía periódicamente su posición a UE mediante sockets UDP. En PowerDEVS un atómico `SendUDP` (Figura 7b) confecciona una trama de datos a partir de las posiciones de la trayectoria con formato:

$$\texttt{dataEsfera} = \langle \texttt{<pos\_x>}, \texttt{<pos\_y>}, \texttt{<pos\_z>} \rangle.$$

En consecuencia, UE actualiza la posición de la esfera y los gráficos con su evolución temporal (Figura 7a arriba a la derecha). De nuevo, la interfaz de usuario de la aplicación en UE permite al usuario controlar libremente la posición del volumen de control (un cubo). Cada vez que un pájaro cruza este cubo, se detecta un evento y se envía un mensaje a PD con la cantidad de aves detectadas en el cubo y su posición mediante sockets UDP. Además, se actualizan los gráficos del número de pájaros dentro del cubo (Figura 7a arriba a la izquierda). Este mensaje externo a PowerDEVS es mapeado en un evento externo del atómico `ReadUDP` cuyo valor es la trama recibida:

$$\texttt{dataCubo} = \langle \texttt{<cant\_aves>}, \texttt{<pos\_cubo>} \rangle.$$

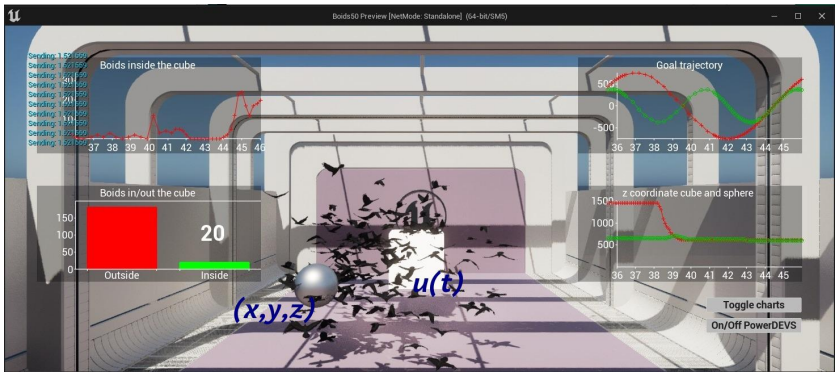
A partir de esta información, el modelo de PD adapta la trayectoria de la esfera en tiempo de simulación para acercarla al cubo. Como resultado, la esfera se traslada (en forma de 8) "buscando acercarse" al volumen de control, maximizando la cantidad de pájaros dentro del mismo a cada momento. Esta comunicación bidireccional sigue la misma lógica que se observa en la Figura 6c para el ejemplo anterior, manteniendo idénticas las partes delimitadas en celeste pero utilizando los datos recibidos para establecer la nueva posición de la esfera, y generando la trama de datos a enviar a partir de la acción de las aves dentro del cubo.

5 Conclusiones y Próximos Pasos

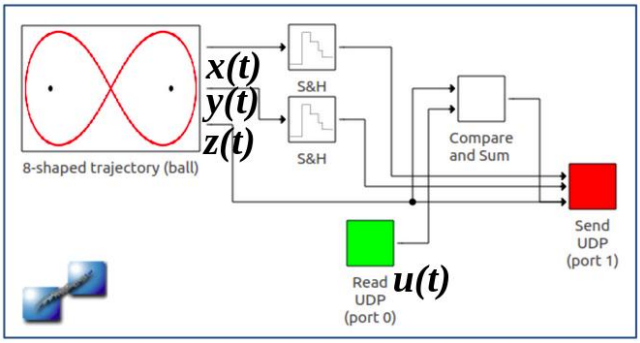
Partiendo de la problemática de generar visualizaciones eficaces y narrativas inmersivas para el análisis de datos reales y generados por simulaciones, con el objetivo de favorecer la toma de decisiones informadas en contextos diversos, introducimos el concepto de *simulación interactiva*, donde el usuario puede explorar parámetros en tiempo real a medida que la simulación evoluciona.

Presentamos un framework que combina el simulador PowerDEVS con el motor gráfico Unreal Engine, mediante un esquema de comunicación basado en sockets UDP. Los casos de estudio descritos nos han mostrado que la combinación de estas herramientas constituye un marco potente para ayudar a las partes interesadas a analizar distintos escenarios, aprovechando nuestra pantalla a gran escala. Para cada nuevo caso de uso será necesario desarrollar sus respectivos modelos de simulación y escenas de animación, preservando los bloques de comunicación en ambas herramientas. De esta manera, el framework propuesto asegura que ambas plataformas puedan interactuar en tiempo real.

Como próximos pasos, se estudiarán alternativas para adaptar y mejorar la cantidad y el tipo de datos que componen la trama de datos enviados, que hoy



(a) Captura de Unreal Engine.



(b) Captura de PowerDEVS.

Fig. 7: Bandada de aves. (a) Animación realista de un modelo de agentes del comportamiento de una bandada de aves en UE, (a) Trayectoria adaptativa en forma de 8, en PD.

se encuentran limitados por la implementación del plugin *UDP Communication*. A su vez, buscamos ampliar el abanico de casos de aplicación, incorporando modelos de sistemas de diversas áreas (sociales, económicos, biológicos, etc.) para problemas relevantes que se beneficien de la interacción inmersiva a la hora del análisis y la toma decisiones.

References

Bergero, F., & Kofman, E. (2011). PowerDEVS: A tool for hybrid system modeling and real-time simulation. *Simulation*, *87*(1-2), 113–132.

Castro, R., Kofman, E., & Wainer, G. (2010). A Formal Framework for Stochastic Discrete Event System Specification Modeling and Simulation. *SIMULATION*, *86*(10). https://doi.org/10.1177/0037549709104482

Castro, R., Bergonzi, M., Marcosig, E. P., Fernández, J., & Kofman, E. (2024). Discrete-event simulation of continuous-time systems: Evolution and state of the art of quantized state system methods. *SIMULATION*, *100*(6), 613–638. https://doi.org/10.1177/00375497241230985

Cesium. (2024). Cesium for unreal.

Chandler, T., Cordeil, M., Czauderna, T., Dwyer, T., Glowacki, J., Goncu, C., Klapperstueck, M., Klein, K., Marriott, K., Schreiber, F., & Wilson, E. (2015). Immersive analytics. *2015 Big Data Visual Analytics (BDVA)*, 1–8. https://doi.org/10.1109/BDVA.2015.7314296

Cui, W. (2019). Visual analytics: A comprehensive overview. *IEEE access*, *7*, 81555–81573.

Epic Games. (2025). Unreal engine site. Retrieved April 9, 2025, from https://www.unrealengine.com

Matković, K., Gračanin, D., & Hauser, H. (2018). Visual analytics for simulation ensembles. *2018 Winter Simulation Conference (WSC)*, 321–335.

Pecker-Marcosig, E., Giribet, J. I., & Castro, R. (2018). Devs-over-ros (dover): A framework for simulation-driven embedded control of robotic systems based on model continuity. *2018 Winter Simulation Conference (WSC)*, 1250–1261. https://doi.org/10.1109/WSC.2018.8632504

Pecker-Marcosig, E., Zudaire, S., Castro, R., & Uchitel, S. (2023). Correct and efficient uav missions based on temporal planning and in-flight hybrid simulations. *Robotics and Autonomous Systems*, *164*, 104404.

SEDLab. (2023). Communication between PowerDEVS and Unreal Engine [Accesed: 30-09-2024].

TalTech. (2023). UDP-Communication for UE4.

Van Tendeloo, Y., & Vangheluwe, H. (2017). An evaluation of devs simulation tools. *Simulation*, *93*, 103–121. https://journals.sagepub.com/doi/full/10.1177/0037549716678330

Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations* (3rd). Academic Press.