Diseño e Implementación de una Herramienta de Software para el Modelado y Simulación en RDEVS (WIP)

María Julia Blas[1,2], Fidel Dalmasso[2], Mateo Toniolo[2], Silvio Gonnet[1,2]
[1]Instituto de Desarrollo y Diseño INGAR (UTN-CONICET)
[2]Universidad Tecnológica Nacional-Facultad Regional Santa Fe
mariajuliablas@santafe-conicet.gov.ar, fideldalmasso@gmail.com,
mateojustotoniolo@gmail.com, sgonnet@santafe-conicet.gov.ar

Resumen

En los últimos años, el campo de Modelado y Simulación ha comenzado a abordar nuevas propuestas basadas en la construcción de modelos conceptuales como soporte al diseño e implementación de modelos de simulación. Bajo esta perspectiva, la definición de un modelo de simulación puede formularse partiendo de un modelo conceptual en el cual se abstrae el dominio del problema bajo estudio. En este trabajo se presenta la arquitectura de una herramienta de software para la construcción, ejecución y visualización de modelos de simulación basados en Routed DEVS. El formalismo Routed DEVS ha sido presentado con anterioridad como una subclase del formalismo Discrete Event System Specification que busca dar solución a la identificación de eventos como funcionalidad embebida dentro de los modelos de simulación. Se presenta un conjunto de módulos de software que toman como punto de partida una representación gráfica del dominio y generan el código Java asociado a los modelos requeridos. Estos módulos son acompañados de módulos de captura de datos y visualización de información con el objetivo de proveer al usuario, de forma simple y amigable, una evaluación del estado resultante de cada corrida de simulación.

1. Introducción

El formalismo Discrete Event System Specification (DEVS) es un formalismo de modelado basado en la teoría de sistemas que proporciona una metodología general para la construcción jerárquica de modelos de simulación reutilizables de forma modular [1]. El núcleo de DEVS incluye un *framework de Modelado y Simulación* (M&S) estructurado en base a tres componentes: *modelo*, *simulador* y *marco experimental*. Cada uno de estos componentes debe ser visto como una entidad independiente que actúa con el objetivo de dar lugar al proceso de M&S. En este sentido, el *modelo* describe la especificación del sistema según su comportamiento y estructura. Por su parte, el *simulador* refiere al sistema computacional que se encarga de ejecutar las instrucciones del modelo. Finalmente, el *marco experimental* representa las condiciones bajo las cuales el sistema es observado, dando lugar al contexto de experimentación y validación del modelo.

Varios autores han mejorado las capacidades de DEVS en respuesta a diferentes situaciones, brindando soluciones aplicables a distintos problemas de simulación asociados a múltiples dominios. Entre las extensiones más populares se encuentran Cell-DEVS [2], Dynamic Structure DEVS [3], Fuzzy-DEVS [4], Min-Max DEVS [5], Parallel DEVS [6], and Vectorial DEVS [7]. En este contexto, Routed DEVS (RDEVS) [8] ha sido desarrollado recientemente como una nueva extensión del formalismo DEVS que facilita el modelado y simulación de problemas centrados en el ruteo de eventos sobre modelos de simulación discretos.

Tanto DEVS como sus extensiones, son formalismos abstractos para la especificación de modelos de simulación de forma independiente de cualquier implementación particular. No obstante, existe una necesidad de que estos modelos sean ejecutables. Por naturaleza, la simulación es un campo técnico [9]. Luego, cuando los ingenieros desean simular modelos formalizados en DEVS, deben programarlos explícitamente en el lenguaje admitido por un simulador específico; lo que, por ejemplo, significa escribir código Java o C++ u otro lenguaje de programación de propósito general [10]. Esta implementación se denomina "reducción a forma concreta" [1].

Sin embargo, cuando se definen implementaciones para modelos DEVS utilizando lenguajes de propósito general es difícil garantizar que los modelos resultantes se ajusten a su formalización [11]. Las propiedades y restricciones matemáticas definidas en los modelos DEVS deben garantizarse en cualquier implementación de los mismos. Por lo tanto, la tarea de diseñar e implementar modelos de simulación basados en DEVS no es trivial. En este contexto, es fundamental contar con entornos de M&S que asistan al modelador en la formalización de los modelos, pero que también posibiliten la obtención de sus implementaciones de forma semiautomática.

Hoy en día, existen múltiples herramientas de software y simuladores para modelos DEVS [12]. Muchas de estas

herramientas admiten capacidades de modelado gráfico. Por ejemplo, PowerDEVS [13] integra diferentes módulos de software con el objetivo de proporcionar un entorno de modelado gráfico, un editor de modelo atómico y un generador de código. En la misma dirección, el entorno de modelado gráfico CD ++ Builder [14] brinda la posibilidad de crear modelos para CD ++ [2]. DEVSimPy [15] ofrece un entorno de modelado gráfico para modelos acoplados. Un enfoque similar es aplicado en Virtual Laboratory Environment [16], donde los modelos atómicos se escriben en C ++ y los modelos acoplados se pueden crear usando el entorno gráfico o escribiendo manualmente archivos XML.

Sin embargo, aunque todas estas herramientas son válidas, estos enfoques se centran en construir gráficamente modelos que ya han sido diseñados por algún modelador o, por el contrario, requieren de un cierto nivel de codificación para obtener las implementaciones finales de los modelos atómicos. Luego, debido al uso de lenguajes de propósito general, frecuentemente las herramientas de M&S se encuentren orientadas al desarrollador.

En los últimos años, el área de M&S ha comenzado a abordar propuestas basadas en la construcción de modelos conceptuales como soporte al diseño e implementación de distintos tipos de modelos de simulación, donde el modelo conceptual es independiente del código del modelo a simular [9]. Desde la perspectiva del modelado, un modelo de simulación es similar a un modelo de sistema de software. Esto se debe a que ambos tipos de modelos se desarrollan a partir de un modelo de sistema conceptual [17-18]. Luego, el modelo conceptual describe el modelo de simulación de forma independiente a los lenguajes de implementación utilizados para darle soporte. En esta dirección, se han desarrollado distintas propuestas con el objetivo de determinar cómo las técnicas de Ingeniería de Software pueden ayudar al modelado conceptual, a la construcción de simulaciones basadas en dicho modelo, y a su validación [9]. En este contexto, una herramienta de software de M&S debe, al menos, proporcionar dos conjuntos distintos de módulos: módulos de software para la definición de los modelos y módulos de software para la ejecución de los modelos de simulación definidos.

En este trabajo se propone una herramienta de software para M&S implementada como un complemento del entorno de desarrollo Eclipse [19] que permite especificar modelos de simulación de procesos de enrutamiento pensados en su abstracción a un modelo de grafos. Como mecanismo de formalización para los modelos de simulación se utiliza el formalismo RDEVS [8]. En este sentido, se presenta la arquitectura de software que da soporte a los diferentes módulos que componen la herramienta final; centrando la definición en tres tipos de módulos: *módulos de especificación de modelos*, *módulos de ejecución de simulación y captura de datos*, y *módulos de visualización de resultados*. De esta manera, el objetivo final de la herramienta bajo desarrollo es: *i)* ofrecer un entorno gráfico para el modelado del formalismo RDEVS utilizando una descripción gráfica estandarizada, y *ii)* generar código Java para estos modelos de forma tal que puedan ser ejecutados haciendo uso de simuladores DEVS.

El resto del trabajo se encuentra estructurado de la siguiente manera. La Sección 2 presenta el formalismo RDEVS, describiendo las formalizaciones de cada uno de los modelos propuestos en la extensión DEVS junto con su interpretación y uso. La Sección 3 describe la herramienta de M&S propuesta, incluyendo una descripción de cada uno de los módulos de software bajo desarrollo y las tecnologías Java que le dan soporte. Finalmente, la Sección 4 se encuentra dedicada a las conclusiones y trabajos futuros.

2. Formalismo Routed DEVS (RDEVS) como Extensión de DEVS

El formalismo RDEVS ha sido presentado en [8] como una subclase del formalismo DEVS [1] que busca dar solución a la identificación de eventos como funcionalidad embebida dentro de modelos de simulación basados en eventos discretos.

Frecuentemente, el objetivo de los modelos de simulación DEVS plantea la necesidad de identificar el origen y/o indicar el destino de los eventos a fin de garantizar su correcto procesamiento. Este problema se conoce como *problema de ruteo de eventos* (o, simplificado, *problema de ruteo*). Un *problema de ruteo* afecta el diseño del modelo de simulación, pero no se encuentra específicamente vinculado al comportamiento de los componentes a diseñar. Es decir, a nivel del escenario a modelar (*dominio*), los componentes no presentan una característica propia de ruteo de mensajes. Sin embargo, a nivel del modelo de simulación (*diseño*), los modelos que representan los distintos componentes requieren funcionalidad de ruteo para los eventos a intercambiar. En este contexto, RDEVS actúa como una capa sobre DEVS que provee funcionalidad de ruteo sin necesidad de que el modelador recurra a nuevas especificaciones DEVS para lograr estas funcionalidades [2]. De esta forma, la incorporación de dicha funcionalidad como parte del formalismo de simulación, reduce la complejidad de diseño ya que el modelador no debe incorporar nuevos módulos de simulación para el manejo de las rutas. Esto es, RDEVS actúa como una *abstracción* entre el *dominio* y el *diseño* que ayuda a generar modelos de simulación reutilizables reduciendo el tiempo de trabajo requerido para la obtención del modelo de simulación final.

El formalismo RDEVS define tres tipos de modelos de simulación, a saber: *modelo esencial*, *modelo de ruteo* y *modelo de red*. En el apartado 3.1 se presentan las definiciones formales de los modelos de simulación propuestos en el formalismo RDEVS, los cuales son interpretados en el apartado 3.2. Teniendo en cuenta que cada *modelo* representa un nivel de *abstracción* utilizado para conceptualizar los elementos que conforman un *problema de ruteo*, en el apartado 3.3 se presentan las relaciones entre los elementos que forman un problema de ruteo (*dominio*), los modelos RDEVS (*abstracción*) y los modelos DEVS (*diseño*). Estas relaciones sientan las bases para la aplicación del formalismo RDEVS en la resolución de un *problema de ruteo*.

2.1. Modelos de Simulación RDEVS

Un *modelo esencial* representa el *comportamiento de un componente*. A fin de garantizar funcionalidad de ruteo sobre modelos DEVS, este modelo queda definido como un modelo DEVS atómico [1]. Luego, un *modelo esencial* es formalmente definido por la estructura

$$E = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau >$$

donde:

$X \equiv$ conjunto de eventos de entrada,

$S \equiv$ conjunto de estados secuenciales,

$Y \equiv$ conjunto de eventos de salida,

$\delta_{int}: S \rightarrow S \equiv$ función de transición interna,

$\delta_{ext}: Q \times X \rightarrow S \equiv$ función de transición externa, en la cual:

$Q = \{(s,e) \mid s \in S, 0 \leq e \leq \tau(s)\} \equiv$ conjunto de estados totales,

$e \equiv$ tiempo transcurrido desde la última transición,

$\lambda: S \rightarrow Y \cup \emptyset \equiv$ función de salida,

$\tau: S \rightarrow \mathbb{R}^{+}_{0,\infty} \equiv$ función de avance de tiempo.

Sobre un *modelo esencial*, se definen uno o más *modelos de ruteo*. Un *modelo de ruteo* representa un nodo que define la *estructura de un componente* como una entidad que posee la capacidad de definir el origen/destino de sus eventos de entrada/salida. Luego, la *estructura de un componente* queda definida como el *comportamiento del componente* junto con su *política de ruteo*. Es decir, al relacionar una *política de ruteo* con un *modelo esencial*, y proveer una *estructura* adecuada para su funcionamiento ensamblado, se obtiene un *modelo de ruteo*. En este sentido, distintos *modelos de ruteo* pueden compartir un mismo *comportamiento* (es decir, un mismo *modelo esencial*).

Luego, un *modelo de ruteo* queda definido formalmente por la estructura

$$R = < \omega, E, M >$$

donde:

$\omega = (u, W, \delta r) \equiv$ *política de ruteo*, en la cual:

$u \in N_0 \equiv$ identificador de la entidad,

$W = \{w_1, w_2, \ldots, w_p \mid w_1, w_2, \ldots, w_p \in N_0\} \equiv$ identificadores de *modelos de ruteo* que representan remitentes habilitados,

$\delta r: S_M \rightarrow T \equiv$ función de ruteo encargada de dirigir eventos de salida, en la cual:

$S_M \equiv$ conjunto de estados del modelo M,

$T = \{t_1, t_2, \ldots, t_k \mid t_1, t_2, \ldots, t_k \in N_0\} \equiv$ identificadores de *modelos de ruteo* que definen posibles destinos,

$E = < X_E, S_E, Y_E, \delta_{int,E}, \delta_{ext,E}, \lambda_E, \tau_E > \equiv$ *modelo esencial* embebido en *R*,

$M = <X_M, S_M, Y_M, \delta_{int,M}, \delta_{ext,M}, \lambda_M, \tau_M> \equiv$ modelo DEVS atómico que describe el funcionamiento de *R*, en el cual:

$X_M = \{(x, h, T) \mid x \in X_E, h \in N_0, T = \{t_1, t_2, \ldots, t_k \mid t_1, t_2, \ldots, t_k \in N_0\}\} \equiv$ conjunto de eventos de entrada identificados, con:

$x \equiv$ valor de entrada definido en E,

$h \equiv$ identificador del modelo remitente,

$T \equiv$ identificadores de modelos destinatarios,

$S_M = S_E \equiv$ conjunto de estados secuenciales,

$Y_M = \{(y, h, T) \mid y \in Y_E, h \in N_0, T = \{t_1, t_2, \ldots, t_k \mid t_1, t_2, \ldots, t_k \in N_0\}\} \equiv$ conjunto de eventos de salida identificados, con:

$y \equiv$ valor de salida definido en E,

$h = u \equiv$ identificador del modelo remitente,

$T \equiv$ identificadores de los modelos destinatarios,

$\delta_{int,M}: S_M \rightarrow S_M = \delta_{int,E} \equiv$ función de transición interna,

$\delta_{ext,M}: Q_M \times X_M \rightarrow S_M \equiv$ función de transición externa, en la cual:

$Q_M = \{(s,e) \mid s \in S_M, 0 \leq e \leq \tau_M(s)\} \equiv$ conjunto de estados totales,

$e \equiv$ tiempo transcurrido desde la última transición,

cuya definición acepta eventos de entrada únicamente cuando se satisface una de las siguientes condiciones: *i)* el evento ha sido enviado a *R* desde un *modelo de ruteo* origen habilitado, *ii)* el evento proviene de una fuente externa, o *iii)* la configuración del modelo lo fuerza a aceptar todos los eventos de entrada. Luego, su definición es:

$$\delta_{ext,M}(s,e,x') = \begin{cases} \delta_{ext,M}(s,e\_c+e,x) & \text{si se cumple } ① \\ s & \text{en otro caso } ② \end{cases}$$

donde:

① $(u \in T \wedge h \in W) \vee (h = 0 \wedge u \in T) \vee (u = 0 \wedge W = \emptyset)$ con $x' = (x, h, T)$ con $e\_c = 0$ luego de la ejecución,

② actualizando el tiempo transcurrido acumulado como $e\_c = e\_c + e$,

$\lambda_M: S_M \rightarrow Y_M \cup \emptyset \equiv$ función de salida que genera eventos identificados, la cual queda definida como: $\lambda_M(s) = (\lambda_E(s), u, \delta_r(s))$

$\tau_M: S_M \rightarrow \mathbb{R}^{+}_{0,\infty} \equiv$ función de avance de tiempo.

Finalmente, un *modelo de red* representa el grafo sobre el cual se define el proceso de ruteo. Dicho proceso queda delimitado por un conjunto de entidades (es decir, *modelos de ruteo*) y sus conexiones. En este caso, con el objetivo de

dejar la determinación de las rutas a las políticas de ruteo asociadas a cada nodo, los *modelos de ruteo* son forzosamente conectados con acoplamientos todos contra todos (excepto si mismos). Además, teniendo en cuenta que el proceso de ruteo se da dentro del *modelo de red*, los eventos que entran/salen del modelo no contienen identificación de origen/destino. Luego, para lograr un correcto funcionamiento del proceso de ruteo, la definición del modelo de red incluye dos funciones de traducción (función de traducción de entradas y función de traducción de salidas) que actúan como nexo entre los valores de entrada/salida y los *modelos de ruteo* que componen el *modelo de red*.

Formalmente, un *modelo de red* queda definido por la estructura

$$N = < X, Y, D, \{ R_d \}, \{ I_d \}, \{ Z_{i,d} \}, T_{in}, T_{out}, Select >$$

donde:

$X \equiv$ conjunto de valores de entrada,

$Y \equiv$ conjunto de valores de salida,

$D \equiv$ identificadores de las entidades que componen la red, con $d \in N_0, \ \forall d \in D$,

Para cada $d \in D$, R_d es un *modelo de ruteo* definido como $R_d =< \omega_d, E_d, M_d >$ con $u_d = d$,

Para cada $d \in D \cup \{ N \}$, I_d es el conjunto de influyentes de d, el cual es definido como $I_d = \{ i \mid i \in D \wedge i \neq d \} \cup \{ N \}$ para mantener los acoplamientos todos-contra-todos (excepto si mismos) dentro de la definición del *modelo de red*,

Para cada $i \in I_d$, $Z_{i,d}$ es la función de traducción entre los eventos de salida del *modelo de ruteo* i y los eventos de entrada del *modelo de ruteo* d, siendo:

$Z_{i,d} = T_{in}$ si $i = N$,

$Z_{i,d} = T_{out}$ si $d = N$,

$Z_{i,d}: Y_{M,i} \rightarrow X_{M,d}$ si $i \neq N \wedge d \neq N$,

$T_{in}: X \rightarrow \{(x, h, T) \mid x \in X, h \in N_0, T = \{ t_1, t_2, \ldots, t_k \mid t_1, t_2, \ldots, t_k \in N_0 \}\} \equiv$ función de traducción de entrada la cual toma un evento de entrada externo y devuelve un evento de entrada identificado dentro del proceso de ruteo (indicando los identificadores de los *modelos de ruteo* a ser utilizados como destinatarios para dicho evento), en la cual:

$x \equiv$ evento de entrada,

$h = 0 \equiv$ identificador del *modelo de ruteo* remitente (el valor 0 indica que el evento proviene de una fuente externa),

$T \equiv$ identificadores de destinarios del evento de entrada,

$T_{out}: \{(y, h, T) \mid y \in Y, h \in N_0, T = \{ (t_1, t_2, \ldots, t_k) \mid t_1, t_2, \ldots, t_k \in N0 \}\} \rightarrow Y \equiv$ función de traducción de salida la cual toma un evento de salida identificado dentro del proceso de ruteo (cuyo destinatario indica que debe ser enviado al exterior del *modelo de red*) y devuelve un evento de salida a ser enviado hacia el exterior de la red, en la cual:

$y \equiv$ evento de salida,

$h \equiv$ identificador del modelo remitente,

$T = \emptyset \equiv$ identificadores de destinatarios (el conjunto vacío indica que el destino es un modelo externo a la red),

$Select: 2D \rightarrow D \equiv$ función desempate para transiciones simultaneas.

2.2. Interpretación de Modelos RDEVS

Sea N un *modelo de red* que representa un sistema sobre el cual se debe resolver un proceso de ruteo. Cuando se produce el arribo de un valor x ($x \in X$) a N, se ejecuta su función de traducción de entrada T_{in} a fin de transformar el valor recibido x en un evento con identificación x' dentro de la red. Al llevar información de destino, el evento x' será propagado a todos los *modelos de ruteo* R_d que componen a N. Cada modelo R_d será el encargado de definir cómo tratar el evento x'.

Cuando un *modelo de ruteo* recibe un evento de entrada x', ejecuta su función de transición externa $\delta_{ext,M}$. Esta función, además de conocer el evento que arriba (es decir, x'), recibe información del estado actual del modelo (denominado s) y del tiempo transcurrido desde su última transición (valor de e). De acuerdo a su definición, la función puede provocar dos clases de cambios de estado en el *modelo de ruteo*. Si el evento cumple con ①, el *modelo de ruteo* evolucionará de acuerdo al cambio de estado definido en la función de transición externa asociada al *modelo esencial* que define su comportamiento (es decir, la función $\delta_{ext,E}$). En otro caso, el *modelo de ruteo* se mantendrá en el estado s (es decir, ignorará el evento x' que ha arribado ya que su *política de ruteo* no le permitió aceptarlo).

De forma independiente al arribo de eventos externos, un *modelo de ruteo* puede cambiar su estado a causa del paso del tiempo. Por definición, si no se da ningún evento externo que altere el estado actual s, el modelo permanecerá en el estado s durante la cantidad de instantes de tiempo definida en la función avance de tiempo para dicho estado (es decir, $\tau(s)$). Cuando el tiempo transcurrido en el estado s (el cual queda indicado como e) es igual a la cantidad de instantes de tiempo que el modelo debe mantenerse en dicho estado (es decir, $e = ta(s)$), el modelo producirá un cambio de estado. Previo al cambio de estado, el *modelo de ruteo* debe producir una salida. Para esto, el modelo ejecuta su función de salida λ_M a fin de generar un evento con identificación que será enviado a destinatarios específicos (es decir, el conjunto de *modelos de ruteo* habilitados en su *política de ruteo)*. El evento de salida y' que se genera en esta instancia, contiene un valor de salida y que se obtiene de la definición de comportamiento asociada al *modelo de*

ruteo (es decir, de la función de salida λ_E que pertenece al *modelo esencial* embebido en el nodo). Una vez producida la salida, el *modelo de ruteo* cambia de estado siguiendo la definición de su función de transición interna $\delta_{int,M}$. Teniendo en cuenta que este cambio de estado obedece al *comportamiento* definido para el nodo (es decir, a su *modelo esencial*), la definición de la función $\delta_{int,M}$ es equivalente a la definición de la función de transición interna de E (es decir, $\delta_{int,E}$).

Si en algún momento se produce un evento con identificación *y'* cuyo destinatario indique que debe ser enviado al exterior de la red, el *modelo de red* ejecuta su función de traducción de salida T_{out}. En este caso, la función actúa en sentido opuesto a la función de traducción de entrada. Por esto, su objetivo es remover la información de ruteo (es decir, los componentes *h* y *T*) a fin de propagar el valor de salida *y* (que compone a *y'*) hacia el exterior del modelo.

2.3. Uso de Modelos RDEVS

El núcleo de RDEVS consiste en la abstracción y organización del flujo de eventos entre modelos de forma independiente a la definición del *comportamiento de los componentes*. De acuerdo con esta perspectiva, el conjunto de componentes requeridos como parte de la simulación puede ser modelado en DEVS y, posteriormente, se puede aplicar el formalismo RDEVS sobre estos modelos a fin de diseñar y simular el *problema de ruteo*. Con el objetivo de clarificar las relaciones entre los distintos niveles de aplicación (*dominio*, *abstracción* y *diseño*), la Figura 1 presenta un esquema de ejemplo.

Como puede observarse, a nivel de *dominio*, un *problema de ruteo* queda definido por un conjunto de elementos conectados de forma selectiva según un conjunto de rutas predefinidas. En el ejemplo (nivel superior de la Figura 1), se definen dos rutas. Cada ruta corresponde a un color (azul y rojo). En este caso, la ruta señalada con color azul marca el camino E2→E1→E2→E3. Por su parte, la ruta indicada con color rojo refiere al camino E2→E1→E2→E4→E3. Tal como se presenta en este problema, un mismo elemento puede repetirse en diferentes lugares de la estructura. Sin embargo, aunque el elemento se encuentre replicado dentro de las rutas definidas, todas las réplicas de un mismo elemento cumplen la misma función (es decir, exhiben el mismo comportamiento) sobre diferentes caminos.

En este contexto, la forma en la cual RDEVS abstrae estos elementos del problema original se presenta en el nivel intermedio (*abstracción*). Como puede observarse, todo el *proceso de ruteo* es modelado en base a un único *modelo de red*. Cada nodo del proceso, se presenta como un *modelo de ruteo*. Estos modelos se encuentran vinculados por acoplamientos todos contra todos (excepto si mismos) a fin de respetar la definición del formalismo. Luego, cada *modelo de ruteo* estructura una entidad específica dentro del proceso integrando un *modelo esencial* con su *política de ruteo*.

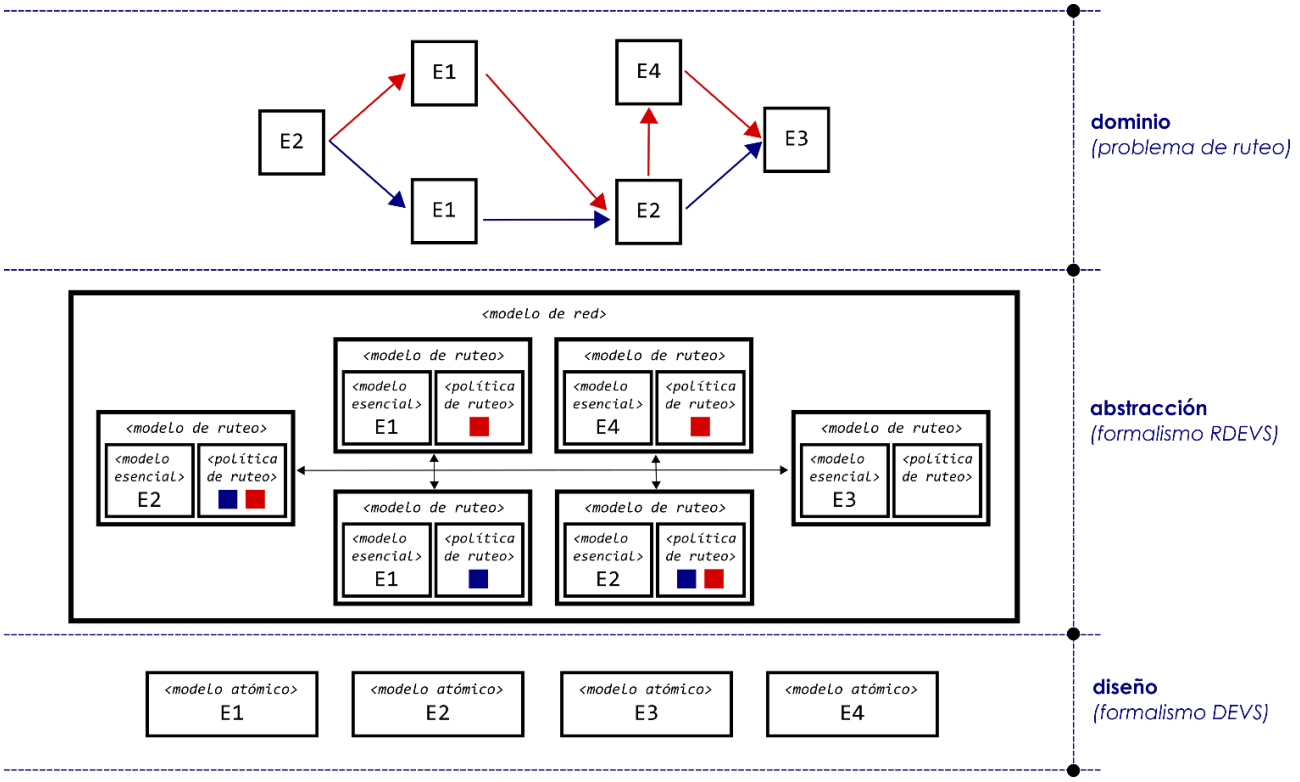

Figura 1. Niveles de aplicación (dominio, abstracción y diseño).

En el nivel inferior (*diseño*), se encuentran los modelos DEVS. Estos modelos pueden ser *atómicos* o *acoplados*. Un *modelo atómico* "describe el comportamiento autónomo de un sistema de eventos discretos como una secuencia de transiciones deterministas entre estados secuenciales junto con la forma en la cual el sistema reacciona a eventos de entrada externos y cómo genera eventos de salida" [20]. Por su parte, un *modelo acoplado* "describe el sistema como una red de componentes DEVS, los cuales pueden corresponder a modelos atómicos o acoplados" [20]. De los dos tipos de modelos propuestos como parte de formalismo DEVS, RDEVS se aplica sobre *modelos atómicos*. Por su propia definición, un *modelo esencial* es un *modelo atómico*. Luego, cada elemento individual que compone el *problema de ruteo* original puede ser visto como un sistema autónomo a ser modelado con un *modelo atómico*. De esta forma, sobre la definición de comportamiento de los elementos individuales (es decir, el *modelo atómico* que equivale a un *modelo esencial*), el formalismo RDEVS provee una capa de abstracción que facilita la implementación de rutas sobre modelos de eventos discretos definidos en DEVS.

3. Herramienta para M&S RDEVS

Teniendo en cuenta que RDEVS se corresponde con una extensión de DEVS, en [8] se presenta un framework implementado en Java que extiende el conjunto de clases del proyecto DEVSJAVA [21] con el objetivo de brindar soporte a la simulación de modelos RDEVS. DEVSJAVA es una herramienta de M&S implementada en Java que soporta la implementación de modelos de simulación basados en el formalismo DEVS. Luego, haciendo uso del framework implementado, un usuario de DEVSJAVA puede diseñar e implementar modelos RDEVS.

Sin embargo, a fin de proveer funcionalidades específicas para el diseño, implementación, ejecución y visualización de resultados de modelos de simulación definidos en RDEVS; la Figura 2 presenta la arquitectura de una herramienta de software integral para su M&S. Como puede observase, esta herramienta se basa en una estructura de capas, donde la plataforma de ejecución corresponde al entorno de desarrollo Eclipse [19]. Dado que Eclipse es una plataforma cuyas funcionalidades pueden extenderse por medio de la instalación de complementos (es decir, plugings), la arquitectura propuesta incluye un conjunto de complementos útiles para la implementación de los módulos de software requeridos como parte de la herramienta final. De esta manera, la herramienta para el M&S basado en RDEVS, es también implementada como un complemento para Eclipse.

En este contexto, los complementos Eclipse utilizados como soporte son: *Eclipse Modeling Framework (EMF)*, *Sirius*, *Acceleo* y *Google Web Toolkit (GWT)*. Cada módulo de la herramienta es diseñado e implementado haciendo uso de un subconjunto de estos complementos.

EMF es una herramienta que provee un marco de trabajo para la definición de modelos junto con un mecanismo de generación de código para la construcción de herramientas de software basadas en modelos de datos estructurados [22-23]. En el caso de la herramienta de M&S para RDEVS, es utilizado como soporte para la definición e implementación de los metamodelos Ecore usados como descripción fundacional del módulo "Especificación de Modelos". Por su parte, *Sirius* es un proyecto Eclipse que permite la creación de entornos de trabajo gráficos empleando las tecnologías de modelado de la plataforma Eclipse [24]. Este proyecto es usado para el desarrollo de representaciones gráficas vinculadas a los metamodelos Ecore definidos con *EMF*.

Acceleo es una tecnología basada en plantillas que incluye herramientas de autoría para crear generadores de código personalizados [25]. En este sentido, permite producir automáticamente cualquier tipo de código fuente a partir de un modelo de datos en formato Ecore. Luego, los modelos Ecore instanciados de forma gráfica haciendo uso de *Sirius*, son traducidos a código fuente Java haciendo uso de este complemento (como parte del módulo "Especificación de Modelos"). En este punto es importante destacar que el código Java que se produce como resultado de este proceso de traducción se corresponde con las implementaciones de clases Java definidas como parte del *RDEVS Framework*. De esta manera, se producen implementaciones válidas de modelos RDEVS que pueden ser ejecutadas utilizando el simulador *DEVSJAVA* (módulo "Ejecución de Modelos").

Finalmente, GWT es un framework creado por Google que posibilita la creación de distintos tipos de gráficos basados en HTML y JavaScript por medio de la complicación de código Java [26]. En el caso de la herramienta de M&S para RDEVS, este complemento es de utilidad para la generación de distintos tipos de gráficos como parte del módulo "Visualización de Información de Ejecución". Sin embargo, dado que para visualizar información de ejecución es necesario previamente relevarla, se incorpora a la herramienta un módulo "Captura de Datos de Ejecución". Este módulo es diseñado con el objetivo de estructurar los resultados de las simulaciones (basadas en *DEVSJAVA*) de los modelos RDEVS (definidos de acuerdo al *RDEVS Framework*) a fin de facilitar su posterior visualización.

3.1. Especificación de Modelos RDEVS: De Modelos Gráficos a Código Java

Tal como se ha enunciado con anterioridad, muchas herramientas de M&S proveen facilidades gráficas para la construcción de modelos. Normalmente, dado que cada tipo de modelo posee su propia definición, las estrategias de representación difieren según el tipo de modelo de simulación a implementar.

Los modelos RDEVS no son ajenos a esta particularidad. Luego, las secciones 4.1.1 y 4.1.2 presentan las estrategias de representación elegidas para los modelos de simulación RDEVS, dividiendo su aplicación entre el *modelo de red* y sus *modelos de ruteos* asociados, y el *modelo esencial*.

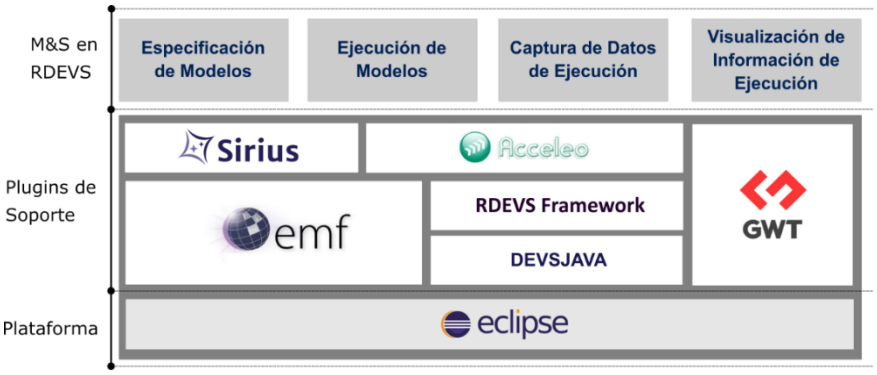

Figura 2. Arquitectura de la herramienta de M&S para RDEVS.

3.1.1. Modelos de Grafos como Especificación de Modelos de Red y Ruteo

Con un alto nivel de abstracción, un *proceso de ruteo* puede ser visto como un grafo, donde los elementos se entienden como nodos y las rutas entre elementos como arcos entre nodos específicos. Al mismo tiempo, un *modelo de red* RDEVS puede ser visto como un grafo ampliado de este grafo inicial; donde los nodos representan *modelos de ruteo* y los arcos representan los acoplamientos entre modelos. Esta representación se considera "ampliada" ya que los arcos del grafo inicial (es decir, las rutas) se mapean a las políticas de ruteo. Luego, los arcos del grafo ampliado representan la estructura de acoplamientos requerida para que el *modelo de red* sea válido.

En este contexto, el módulo "Especificación de Modelos" plantea una representación basada en grafos para la definición del *proceso de ruteo*. Esta representación utiliza un metamodelo Ecore (implementado con *EMF*) que incluye conceptos y relaciones que permiten instanciar modelos de grafos como parte de la definición de *procesos de ruteo*.

Tomando este metamodelo como especificación estructural, se diseñó una representación gráfica para cada componente a instanciar. El objetivo de esta representación es brindar al usuario una herramienta gráfica para la definición del *proceso de ruteo* a simular. De esta manera, el usuario de la herramienta de M&S crea una instancia del metamodelo Ecore (es decir, un modelo de grafo) haciendo uso de una paleta de componentes gráficos. Una vez definido el grafo inicial (es decir, el *proceso de ruteo*), el modelo de grafo es mapeado automáticamente al grafo ampliado requerido para la definición del *modelo de red*.

A partir de este nuevo grafo, el módulo "Especificación de Modelos" genera las clases Java requeridas para dar soporte a la implementación del *modelo de red* asociado al *proceso de ruteo* modelado. Esta implementación incluye la creación de las clases Java asociadas a los *modelos de ruteo* que componen el *modelo de red*. En ambos casos, el proceso de generación de código se codificó utilizando *Acceleo*. Además, como se ha mencionado con anterioridad, las implementaciones obtenidas como resultado de la transformación modelo-a-código se basan en las clases Java incluidas en el *RDEVS Framework*.

A modo de ejemplo, la Figura 3 presenta el modelado del problema de ruteo propuesto en la Figura 1 haciendo uso del módulo "Especificación de Modelos". Como puede observarse, el usuario crea el modelo de su *proceso de ruteo* haciendo uso de los elementos definidos en la paleta de herramientas (situada en la parte derecha de la pantalla). Una vez que el modelo queda completamente definido, el usuario tiene la posibilidad de validar la estructura del grafo. Si el modelo propuesto es válido, se habilita al usuario la opción "Obtener implementación". En caso contrario, se indican los errores encontrados en el modelo actual y se brinda la posibilidad de reformular el modelo a fin de volver a validarlo. Al accionar la opción "Obtener implementación", el usuario generará el código Java asociado a los *modelos de ruteo* y *red* requeridos para la simulación del *proceso de ruteo* modelado.

3.1.2. Diagramas de Estados como Especificación de Modelos Esenciales

Un *modelo esencial RDEVS* es definido como un *modelo atómico DEVS*. Usualmente, este último tipo de modelos se representa de forma gráfica haciendo uso de un diagrama de estados [27]. En este contexto, a fin de especificar el comportamiento de los elementos de ruteo, la herramienta de M&S para RDEVS sigue la misma estrategia de representación que las herramientas de M&S comerciales como, por ejemplo, MS4Me [28].

Sin embargo, a fin de mejorar las representaciones existentes, el diagrama de estados utilizado como base incorpora nociones propias de la definición de modelos atómicos DEVS. De esta manera, el módulo "Especificación de Modelos" incorpora la semántica de los modelos atómicos DEVS (y, en consecuencia, los modelos esenciales RDEVS) como parte de la propia estructura de definición del diagrama. Bajo esta perspectiva, el metamodelo que brinda soporte a la construcción de *modelos esenciales*, incluye conceptos propios del dominio DEVS para dar lugar a la generación del diagrama de estados.

Al igual que en el caso de la representación basada en grafos, la representación basada en diagramas de estados se genera a partir de un metamodelo Ecore (definido con *EMF*). Sobre dicho modelo, se define un entorno gráfico (haciendo uso de *Sirius*) en el cual cada elemento que

compone el diagrama posee su propia representación. Entonces, la forma en la cual un usuario genera una representación gráfica para un *modelo esencial* es similar a la descripta para el caso del *modelo de red* y sus *modelos de ruteo* asociados.

Actualmente se está finalizando la implementación de la instanciación gráfica del modelo asociado al diagrama de estado. El paso siguiente en esta dirección es la generación automática del código Java (basado en *RDEVS Framework*) asociado al modelo de simulación a ejecutar. Para esto, al igual que en el caso previo, se utilizarán las facilidades provistas por *Acceleo* (a fin de transformar la instancia Ecore, diseñada y validada de forma gráfica, en código fuente Java).

3.2. Ejecución de Modelos de Simulación RDEVS

Como se ha indicado con anterioridad, las implementaciones de modelos RDEVS basadas en *RDEVS Framework* han sido definidas siguiendo el conjunto de clases detalladas en el proyecto *DEVSJAVA*. Luego, estas implementaciones pueden ejecutarse haciendo uso del simulador DEVS implementado en dicho proyecto. Adicionalmente, el visualizador de *DEVSJAVA* puede ser utilizado para controlar el proceso de simulación.

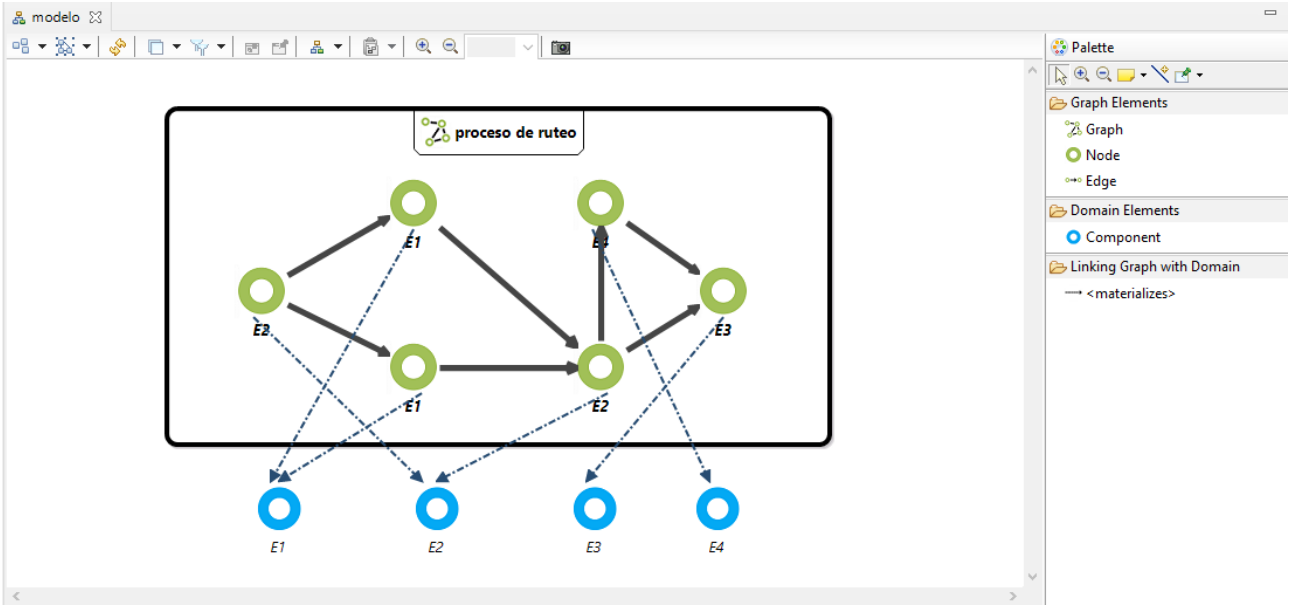

Figura 3. Modelo del proceso de ruteo a ser implementado en RDEVS.

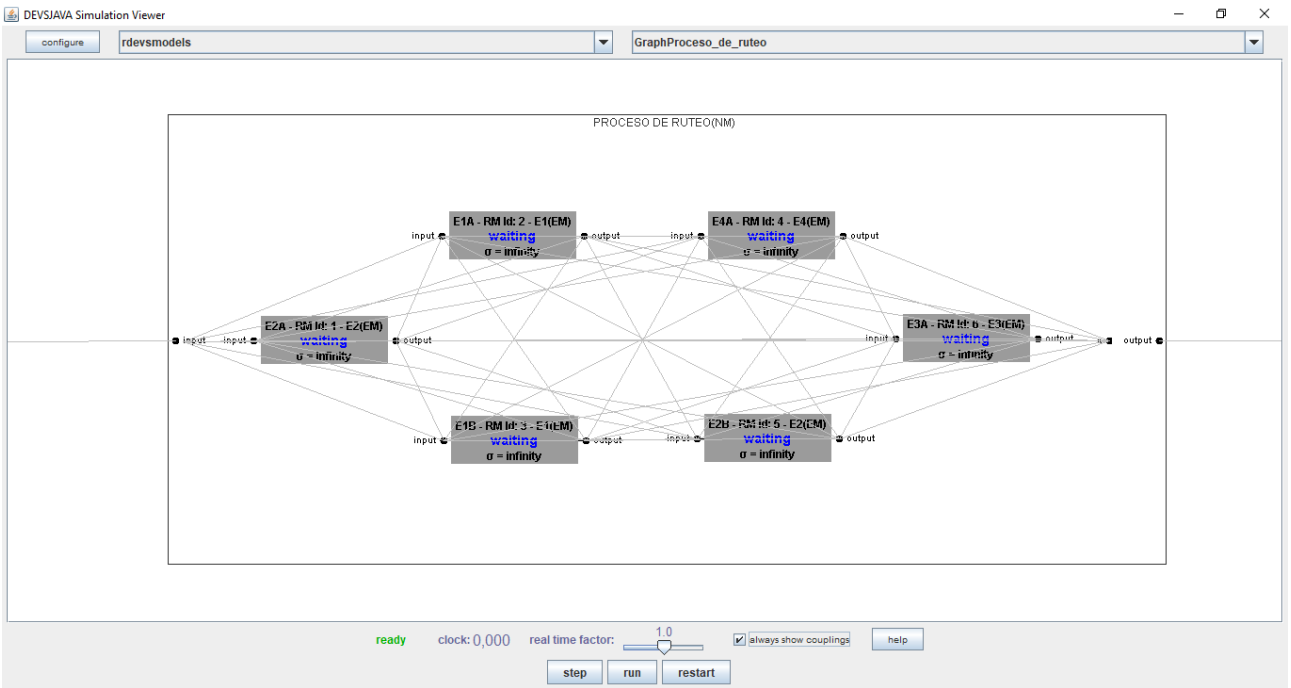

Figura 4. Simulación de un modelo de red RDEVS utilizando DEVSJAVA Viewer.

Bajo esta perspectiva, todos los modelos RDEVS que se generan como parte de las transformaciones modelo-a-código descriptas en la Sección 4.1 son ejecutables desde *DEVSJAVA*. En este sentido, con el objetivo de garantizar la correcta ejecución de las corridas de simulación, el módulo "Ejecución de Modelos de Simulación" se encarga de realizar las vinculaciones requeridas entre las clases que implementan los modelos RDEVS y las clases asociadas a los simuladores DEVS. La Figura 4 presenta el *modelo de red* asociado al *proceso de ruteo* definido en la Figura 3 haciendo uso del visualizador de *DEVSJAVA*.

En este punto, es importante destacar que para cada corrida de simulación es necesario relevar sus datos de estado. Estos datos no se encuentran asociados a las salidas propias del escenario en estudio (ya que esta información debe relevarse como parte de la estructura de los modelos en sí mismos), sino que tienen que ver con el estado de los modelos, eventos y rutas definidos a nivel del formalismo. Por este motivo, este módulo incorpora un conjunto de rastreadores (*trackers*) sobre los modelos de simulación RDEVS a fin de capturar los datos requeridos durante la ejecución de la simulación. Para esto, se hace uso de las estructuras de datos definidas en el módulo "Captura de Datos de Ejecución".

3.3. Captura de Datos de Ejecución: Estados de Modelos, Eventos y Rutas

Un modelo de simulación basado en eventos puede verse desde múltiples puntos de vista. En términos generales, un modelo basado en DEVS puede analizarse desde su vista *estática* (referida a la estructura de estado del dominio) y desde su *vista de comportamiento* (la cual se ocupa de la dinámica del dominio).

La *vista estática* de los modelos RDEVS se corresponde con las representaciones descriptas en la Sección 4.1. Por su parte, la vista dinámica se vincula no sólo con la visualización del modelo durante el proceso de simulación sino también con la forma en la cual el modelo evoluciona a través del tiempo. En este sentido, tal como se ha indicado en la Sección previa, el visualizador de *DEVSJAVA* puede ser utilizado para contemplar la forma en la cual evoluciona el modelo a lo largo del tiempo. Sin embargo, a fin de capturar datos vinculados a dicha evolución, la herramienta de M&S para RDEVS incorpora el módulo "Captura de Datos de Ejecución".

Tal como se ha enunciado en la Sección 4.2, este módulo tiene por objetivo la estructuración de la información que se produce durante el proceso de simulación de modelos RDEVS. Para esto, el módulo incluye un conjunto de clases Java que actúan como rastreadores de puertos, acoplamientos y eventos vinculados a los tres niveles de modelado definidos como parte del formalismo. Es decir, para cada tipo de modelo RDEVS, se define su propio rastreador. Las clases que modelan los rastreadores de los tres niveles se vinculan entre sí a fin de compartir información de estado y garantizar su consistencia. Luego, estos rastreadores almacenan información estructurada a medida que avanza una corrida de simulación.

Para cada nueva corrida de simulación, un nuevo rastreador es utilizado. Al finalizar la ejecución de una corrida de simulación, el módulo "Ejecución de Modelos de Simulación" se encarga de almacenar los datos recopilados en cada uno de los rastreadores utilizados en dicha corrida. Esta información es almacenada en formato JSON para facilitar su posterior manipulación.

3.4. Visualización de Información de Simulación: Gráficas de Estado

Tomando como punto de partida la información almacenada en formato JSON, el módulo de "Visualización de Información de Ejecución" es el encargado de generar reportes. Como en la mayoría de las herramientas de M&S, estos reportes se plantean de forma gráfica a fin de facilitar su interpretación. Sin embargo, a futuro, se podrían incorporar nuevos tipos de reportes.

En base a los registros resultantes del proceso de simulación, los datos son estructurados en nuevos modelos de datos según el tipo de gráfico a visualizar. Los gráficos son generados haciendo uso de *GWT*.

En la actualidad, se está trabajando en la generación de diagramas Sankey. El diagrama de Sankey es un tipo específico de diagrama de flujo en el cual el ancho de las flechas se visualiza de forma proporcional a la cantidad de flujo entre dos nodos específicos. Bajo esta premisa, siendo cada nodo del diagrama un modelo de ruteo, se utiliza el ancho de las flechas para representar el intercambio de eventos entre modelos (diferenciando eventos internos/externos, aceptados/rechazados y contenido del mensaje, entre otros). Para esto, la estructura de datos resultante del proceso de captura, es mapeada a un nuevo modelo estructural en el cual se incluyen conceptos vinculados a este tipo de diagramas.

4. Conclusiones y Trabajos Futuros

En este trabajo se ha presentado la arquitectura de una herramienta de software para el M&S basado en el formalismo RDEVS. El conjunto de módulos de software propuestos como parte de la herramienta provee al usuario un conjunto de funcionalidades para el diseño, implementación y ejecución de modelos RDEVS, destacándose: *i)* facilidad en la especificación de modelos de simulación de eventos discretos que resuelven problemas de ruteo por medio del uso de una abstracción basada en grafos, *ii)* facilidad en la definición de comportamientos DEVS por medio del uso de diagramas de estado, *iii)* ejecución de modelos y captura de información de estado mediante la integración de los módulos RDEVS con el simulador DEVSJAVA, y *iv)* construcción de gráficos para la visualización de información de ejecución una vez finalizada una corrida específica.

En la actualidad, algunos de los módulos ya se encuentran en funcionamiento, habiendo presentado resultados satisfactorios para el M&S de problemas reales. En el caso de los módulos "Especificación de Modelos-

Diagramas de Estados como Especificación de Modelos Esenciales" y "Visualización de Información de Ejecución", aún se está trabajando en su desarrollo. Una vez que estos módulos superen las fases de prueba, el siguiente paso en esta dirección es integrarlos con los módulos ya disponibles en la herramienta.

De esta manera, se garantizará una completa compatibilidad de todos los niveles de modelado, ejecución de corridas de simulación y visualización de resultados que han sido definidos como parte de la herramienta de M&S en RDEVS. El objetivo final de esta línea de trabajo es la construcción de un único complemento para Eclipse basado en DEVSJAVA que brinde solución a la definición, implementación y ejecución de modelos RDEVS a partir del uso de modelos de grafos como esquemas conceptuales de dominio.

Agradecimientos

Los autores agradecen el apoyo brindado por parte de la Universidad Tecnológica Nacional – Facultad Regional Santa Fe por medio de los proyectos de investigación PID SIUTIFE0007638TC y PID UTI5273TC.

Referencias

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

[2] Wainer, G. A., "Modeling and simulation of complex systems with Cell-DEVS", in Proceedings of the 2004 Winter Simulation Conference (WCS'04), December 2004.

[3] Barros, F. J., "Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation", in Proceedings of the 1995 Winter Simulation Conference (WSC'95), December 1995, pp. 781-785.

[4] Kwon, Y., Park, H., Jung, S., and Kim, T., "Fuzzy-DEVS formalism: concepts, realization and applications", in Proceedings of the 1996 Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems, August 1996, pp. 227-234.

[5] Hamri, M. E. A., Giambiasi, N., and Frydman, C., "Min–Max-DEVS modeling and simulation", Simulation Modelling Practice and Theory, 14(7), 2006, pp. 909-929.

[6] Chow, A. C. H., and Zeigler, B. P., "Parallel DEVS: A parallel, hierarchical, modular modeling formalism", in Proceedings of the 1994 Winter Simulation Conference (WSC'94), December 1994, pp. 716-722.

[7] Bergero, F., and Kofman, E., "A vectorial DEVS extension for large scale system modeling and parallel simulation", Simulation, 90(5), 2014, pp. 522-546.

[8] Blas, M. J., Gonnet, S., and Leone, H., "Routing structure over discrete event system specification: A DEVS adaptation to develop smart routing in simulation models", in Proceedings of the 2017 Winter Simulation Conference (WSC'17), December 2017, pp. 774-785.

[9] Robinson, S., "Conceptual modelling for simulation: Progress and grand challenges", Journal of Simulation, 1-20, 2019.

[10] Cristiá, M., Hollmann, D., and Frydman, C., "A multi-target compiler for CML-DEVS", Simulation, 95(1), 2019, pp. 11-29.

[11] Sarjoughian, H. S., Alshareef, A., and Lei, Y., "Behavioral DEVS metamodeling", in Proceedings of the 2015 Winter Simulation Conference (WSC'15), December 2015, pp. 2788-2799.

[12] Van Tendeloo, Y., and H. Vangheluwe, "An evaluation of DEVS simulation tools", Simulation, 93(2), 2017, pp. 103-121.

[13] Bergero F, and Kofman E., "PowerDEVS: a tool for hybrid system modeling and real-time simulation", Simulation, 87(1), 2011, pp. 113–132.

[14] Bonaventura, M., Wainer, G. A., and Castro, R., "Graphical modeling and simulation of discrete-event systems with CD++Builder", Simulation, 89(1), 2013, 4–27.

[15] Capocchi, L., Santucci, J. F., and Poggi, B., "DEVSimPy: a collaborative Python software for modeling and simulation of DEVS systems", in Proceedings of the Workshop on enabling technologies: infrastructure for collaborative enterprises, June 2011, pp.170–175.

[16] Quesnel, G., Duboz, R., and Ramat, E., "VLE: a multimodeling and simulation environment", in Proceedings of the 2007 Summer Simulation Multiconference, July 2007, pp.367–374.

[17] Guizzardi, G., and Wagner, G., "Towards an ontological foundation of discrete event simulation", in Proceedings of the 2010 Winter Simulation Conference (WSC'10), December 2010, pp. 652-664.

[18] Vangheluwe, H., "Foundations of Modelling and Simulation of Complex Systems", in Proceedings of the Seventh International Workshop on Graph Transformation and Visual Modeling Techniques, 2008.

[19] The Eclipse Foundation. *Eclipse*. URL: https://www.eclipse.org/. Accedido por última vez el 1/9/2020.

[20] Vangheluwe, H., The Discrete Event System Specification (DEVS) Formalism, Technical Report, 2001.

[21] Arizona Center for Integrative Modeling and Simulation (ACIMS). *DEVSJAVA*. http://acims.asu.edu/software/devsjava/. Accedido por última vez el 1/9/2020.

[22] The Eclipse Foundation: Eclipse Modeling Project. *Eclipse Modeling Framework*. https://www.eclipse.org/modeling/emf/. Accedido por última vez el 1/9/2020.

[23] Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M., *EMF: Eclipse Modeling Framework*, Pearson Education, 2008.

[24] The Eclipse Foundation. *Sirius*. https://www.eclipse.org/sirius/. Accedido por última vez el 1/9/2020.

[25] The Eclipse Foundation. *Acceleo*. https://www.eclipse.org/acceleo/. Accedido por última vez el 1/9/2020.

[26] Google Web Toolkit. *GWT Project*. http://www.gwtproject.org/. Accedido por última vez el 1/9/2020.

[27] Shaikh, R., and Vangheluwe, H., "Transforming UML2. 0 class diagrams and statecharts to atomic DEVS", in

Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, April 2011, pp. 205-212.

[28] Seo, C., Zeigler, B. P., Coop, R., and Kim, D., “DEVS modeling and simulation methodology with MS4 Me software tool”, in Proceedings of the 2013 Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium, April 2013, pp. 1-8.