Herramienta de Modelado y Generación de Código Java para Simulaciones RDEVS

Modeling and Java Code Generation Tool for RDEVS Simulations

Clarisa Espertino

Universidad Tecnológica Nacional – Facultad Regional Santa Fe
cespertino@frsf.utn.edu.ar

Resumen. En este trabajo se presenta una herramienta de software de modelado textual implementada como un complemento para Eclipse que facilita la definición de procesos de enrutamiento utilizando especificaciones textuales. Estos procesos definen un modelo de simulación que admite la generación de código Java para situaciones de enrutamiento, a fin de que puedan ser ejecutadas en simuladores Discrete Event System Specification (DEVS) como modelos Routed DEVS (RDEVS). Así, haciendo uso de una especificación textual, un modelador no experto en RDEVS podrá obtener modelos de simulación de eventos discretos ejecutables. El núcleo de esta herramienta es una gramática libre de contexto que define la estructura del texto. Además, incluye un editor que permite crear especificaciones textuales basadas en dicha gramática, ofreciendo ayudas de escritura para asistir al usuario durante la edición. Partiendo de una herramienta para crear archivos de especificación adecuados, la herramienta tiene una opción de validación que permite procesar y validar el contenido escrito, para garantizar la consistencia de los modelos a generar. Finalmente, la herramienta genera el código Java para la simulación de los modelos.

Palabras Clave: gramática libre de contexto, modelos de simulación, generación de código, Routed DEVS.

Abstract. This paper presents a textual modeling software tool implemented as a plugin for Eclipse that facilitates the definition of routing processes using textual specifications. These processes define a simulation model that supports the generation of Java code for routing situations, in order to execute them in Discrete Event System Specification (DEVS) as Routed DEVS (RDEVS) models. In this way, using a textual specification, a modeler who is not an expert in RDEVS will be able to obtain executable discrete event simulation models. The core of this tool is a context-free grammar that defines the structure of the text. In addition, it includes an editor that allows to create textual specifications based on the grammar, offering writing aids to assist the modeler during the

Received April 2023 Accepted July 2023; Published July 2023
https://doi.org/10.24215/15146774e024
ISSN 1514-6774

edition. Starting from a tool to create suitable specification files, the tool has a validation option that allows to process and validate the written content, to guarantee the consistency of the models to be generated. Finally, the tool generates the Java code for the models's simulation.

Keywords: context-free grammar, simulation models, code generation, Routed DEVS.

1 Introducción

En la actualidad, muchos modelos de simulación son definidos a través de lenguajes de programación lejanos al formalismo matemático sobre el que están creados. En particular, para el formalismo Routed DEVS (RDEVS) [1] se deben utilizar librerías Java durante el proceso de modelado y simulación (M&S) para obtener modelos ejecutables. Es decir, el modelador debe generar clases Java para definir sus modelos de simulación. Esto puede ser un obstáculo para aquellos modeladores que no cuentan con los conocimientos de programación necesarios. En este contexto, es fundamental contar con entornos de M&S que brinden a los modeladores la posibilidad de especificar y editar modelos en distintos lenguajes (tanto gráficos como textuales), facilitando la obtención de sus contrapartes ejecutables (implementaciones) que luego puedan ser ejecutadas en simuladores concretos.

El formalismo RDEVS ha sido definido en [1] como una extensión del formalismo DEVS [2] que facilita la definición de procesos de enrutamiento en modelos de simulación basados en eventos discretos. Por tratarse de un formalismo propuesto recientemente, es necesario el desarrollo de nuevas herramientas que faciliten su uso, para que no sólo aquellos modeladores que cuenten con las habilidades necesarias puedan generar sus especificaciones a través de RDEVS.

En respuesta a la limitación presentada, en [3] se propone una herramienta de software de modelado gráfico desarrollada como un complemento para Eclipse que facilita la especificación de la estructura de un escenario RDEVS, abstrayendo la definición de procesos de enrutamiento en grafos basados en nodos y enlaces. A través de la herramienta mencionada, es posible obtener modelos de simulación de eventos discretos a partir de la definición gráfica de un modelo de grafo que conceptualiza procesos de enrutamiento, reduciendo tiempos de implementación y asegurando la correctitud de los modelos obtenidos. Una desventaja de su uso es que para estructuras de gran tamaño pueden generarse diagramas extensos que dificulten su entendimiento y edición. Por esta razón, en este trabajo se presenta una herramienta de modelado textual como alternativa a la propuesta presentada en [3]. La misma fue implementada como un complemento para el entorno de desarrollo Eclipse [4], y permite especificar la estructura de procesos de enrutamiento de forma textual.

El núcleo de la herramienta propuesta es una *gramática libre de contexto* acompañada de un *metamodelo* que define la estructura requerida. Una *gramática libre de contexto* es una forma de describir lenguajes mediante reglas recursivas llamadas

producciones. Consta de un conjunto de variables, un conjunto de símbolos terminales y una variable inicial, así como de producciones. Cada producción consta de una variable de cabeza y un cuerpo, formado por una cadena de cero o más variables y/o símbolos terminales [5]. Por otro lado, un *metamodelo* es una herramienta de modelado que permite asegurar la correctitud de la estructura de modelos. Se trata de un modelo que define el lenguaje utilizado para diseñar un modelo [6].

En este contexto, la herramienta presentada se basa en una gramática libre de contexto que ha sido implementada con ANTLR4 [7] y ofrece al modelador una variedad de estructuras permitidas, tanto en inglés como en español. Además, incluye: *i)* un editor de texto que da soporte al uso de la gramática ofreciendo ayudas de escritura, *ii)* un asistente para la creación de archivos que contienen especificaciones estructuradas y poseen una extensión determinada, *iii)* un proceso de validación que deriva en una instancia del metamodelo propuesto, y *iv)* un traductor a modelos RDEVS, el cual permite generar las clases Java asociadas al escenario de simulación especificado, sin la necesidad de codificar en un lenguaje de programación específico. A través de la gramática diseñada, se analiza sintácticamente el contenido creado con el editor de texto. Como parte del desarrollo de la herramienta, se implementó una versión Ecore de un metamodelo de un proceso de enrutamiento. Luego, este metamodelo permite instanciar modelos de procesos de enrutamiento teniendo en cuenta un conjunto de restricciones que garantizan la posterior obtención de los modelos RDEVS equivalentes. Estos modelos corresponden a clases Java.

Este trabajo es una versión extendida de "Herramienta de Modelado Textual como Soporte al Formalismo RDEVS" [8], presentado en el Concurso de Trabajos Estudiantiles (EST 2022) de las Jornadas Argentinas de Informática (JAIIO). Además de incorporar mayor nivel de detalle respecto a los elementos sintácticos utilizados para la construcción de la herramienta, incluye el conjunto de transformaciones abordadas para la obtención de las clases Java asociadas al proceso de enrutamiento definido.

El resto del trabajo se encuentra estructurado de la siguiente manera. La Sección 2 presenta los conceptos de procesos de enrutamiento y modelos de red restringidos, y la relación entre ellos. La Sección 3 presenta la estructura de la herramienta propuesta, describiendo la gramática libre de contexto sobre la que se basa y el metamodelo utilizado para instanciar procesos de enrutamiento. Luego, la Sección 4 describe la herramienta propuesta para definir los procesos de enrutamiento junto con un ejemplo ilustrativo y detalles sobre las validaciones y transformaciones realizadas en la traducción a los modelos RDEVS. Por último, la Sección 5 está dedicada a las conclusiones y trabajos futuros.

2 Procesos de Enrutamiento como una Conceptualización de Modelos de Red Restringidos

Un proceso de enrutamiento puede ser definido como "un sistema de componentes que interactúan, donde la operación de un componente y el ruteo de sus salidas depende de lo que sucede a lo largo del proceso" [9]. Luego, la operación interna de los distintos componentes es independiente de la estructura del proceso que definen. Así, los componentes pueden decidir los destinos de sus salidas y tomar decisiones sobre el ruteo.

En las siguientes secciones se describe brevemente el formalismo RDEVS y la forma en la cual, a través del uso de la herramienta propuesta, se pueden definir los procesos de enrutamiento que son la base para la obtención de modelos de simulación RDEVS.

2.1 Formalismo RDEVS

El formalismo RDEVS surge como una subclase del formalismo DEVS que provee una base sólida para el M&S de procesos de enrutamiento que son estudiados como sistemas de eventos discretos.

Este formalismo estructura tres tipos de modelos: *modelo esencial, modelo de ruteo* y *modelo de red*. Cada modelo cumple un rol en la especificación de procesos de enrutamiento. La Tabla 1 presenta la descripción de cada uno de ellos.

Tabla 1. Descripción de los elementos RDEVS.

Elemento RDEVS	***Descripción***
Modelo Esencial	Funcionamiento de un componente elemental que es tomado como base en la definición de un proceso de enrutamiento.
Modelo de Ruteo	Definición de un modelo esencial junto con una política de ruteo. Un mismo modelo esencial puede componer a múltiples modelos de ruteo dentro de un mismo modelo de red.
Política de ruteo	Formalización de interacciones entrada-salida y salida-entrada vinculadas a los componentes elementales. Estas interacciones alteran el normal funcionamiento de los componentes, permitiéndoles elegir cuáles de los eventos entrantes serán aceptados y, al mismo tiempo, indicar los destinos asociados a los eventos salientes.
Modelo de Red	Conjunto de modelos de ruteo acoplados todos contra todos.

2.2 Procesos de Enrutamiento como Modelos de Red Restringidos para la Obtención de Modelos RDEVS

La teoría de redes propone modelar un sistema como un conjunto de nodos conectados a través de enlaces [10]. Luego, una red consiste en nodos conectados por un conjunto de enlaces. De acuerdo con lo definido en [11], si un proceso de enrutamiento es correctamente definido como un modelo de red restringido, los modelos RDEVS pueden ser obtenidos a partir de un conjunto de reglas de traducción. Es decir, es posible establecer una correspondencia entre los elementos de la teoría de redes con los modelos RDEVS. Así, un *modelo esencial* define el comportamiento o funcionalidad de un *nodo.* Un mismo modelo esencial puede corresponderse con uno o más nodos. Un *modelo de ruteo* también se asocia con el concepto de *nodo,* ya que define su estructura o ubicación dentro de la red. Un *modelo de red* representa un conjunto de nodos conectados todos contra todos, conceptualizando el modelo de *red.* Los acoplamientos todos contra todos son requeridos por la definición del modelo de red, para así permitir que el ruteo se delegue al conjunto de funciones de ruteo. Por último, las *políticas de ruteo* formalizan el conjunto de *enlaces* tanto entrantes como salientes

de un nodo. A partir de esto, se puede establecer que cualquier definición de un modelo de red que cumple con ciertas restricciones para modelar un proceso de enrutamiento puede tomarse como base para la obtención de modelos RDEVS.

En las siguientes dos figuras se presenta un ejemplo representativo para ilustrar el mapeo entre los conceptos de la teoría de redes y los modelos RDEVS. En la Figura 1, se puede observar una red compuesta por 8 nodos (representados con máquinas). Cada nodo se corresponde con un ícono que indica su tipo. Estos tipos refieren a su comportamiento (para este ejemplo, es suficiente con abstraerse de la funcionalidad específica de cada máquina). Además, se puede observar que existen enlaces entre los nodos definidos. Los colores que los identifican representan las diferentes alternativas existentes para las salidas de ciertas máquinas (como las de *MACHINE1* y *MACHINE5)* y las entradas de otras (*MACHINE4*).

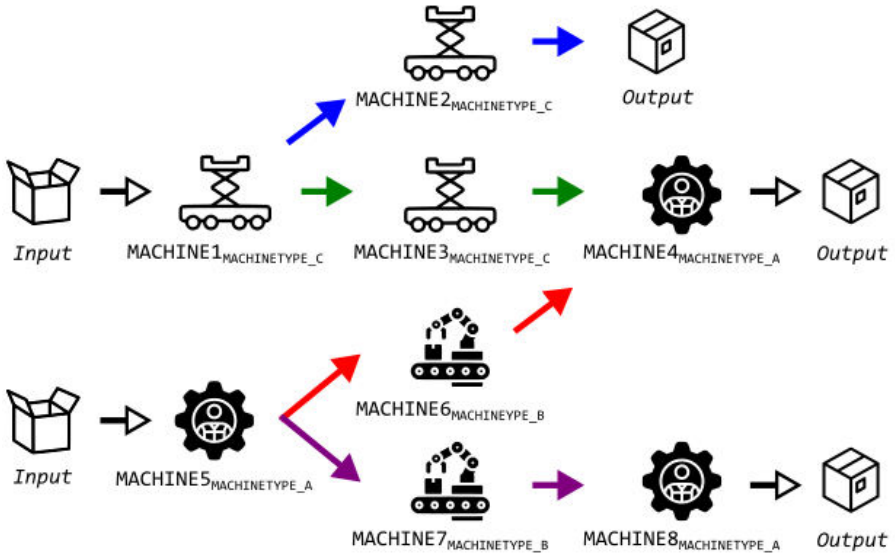

Fig. 1. Esquema de la estructura de una red.

La Figura 2 muestra el modelo asociado a la red presentada utilizando RDEVS. Para construir este esquema, se debe definir un *modelo esencial* (ME_i) para cada tipo de máquina. Para este ejemplo, *MACHINE1, MACHINE2* y *MACHINE3* tendrán el mismo modelo esencial, identificado como ME_3. De igual modo ocurre con *MACHINE4, MACHINE5* y *MACHINE8* (todas poseen a ME_1 como modelo esencial) y con *MACHINE6* y *MACHINE7* (ME_2). Luego, para cada máquina se crea un *modelo de ruteo* (MR_i) y todos los resultantes se conectan todos contra todos (por lo mencionado en párrafos previos sobre la definición del *modelo de red*). Las *políticas de ruteo* (PR_i) se definen teniendo en cuenta los enlaces planteados en el escenario de ejemplo. Así, el modelo de ruteo identificado como MR_1 cuenta con más de una política, representadas en colores verdes y azules para señalar que se asocian con el hecho de que la salida de *MACHINE1* posee dos alternativas (algo similar ocurre con MR_5, con los caminos identificados con colores rojo y violeta). Las políticas señaladas en color amarillo

indican que las máquinas que las contienen se asocian con las salidas del modelo de red (como es el caso de *MACHINE2, MACHINE4* y *MACHINE8)*.

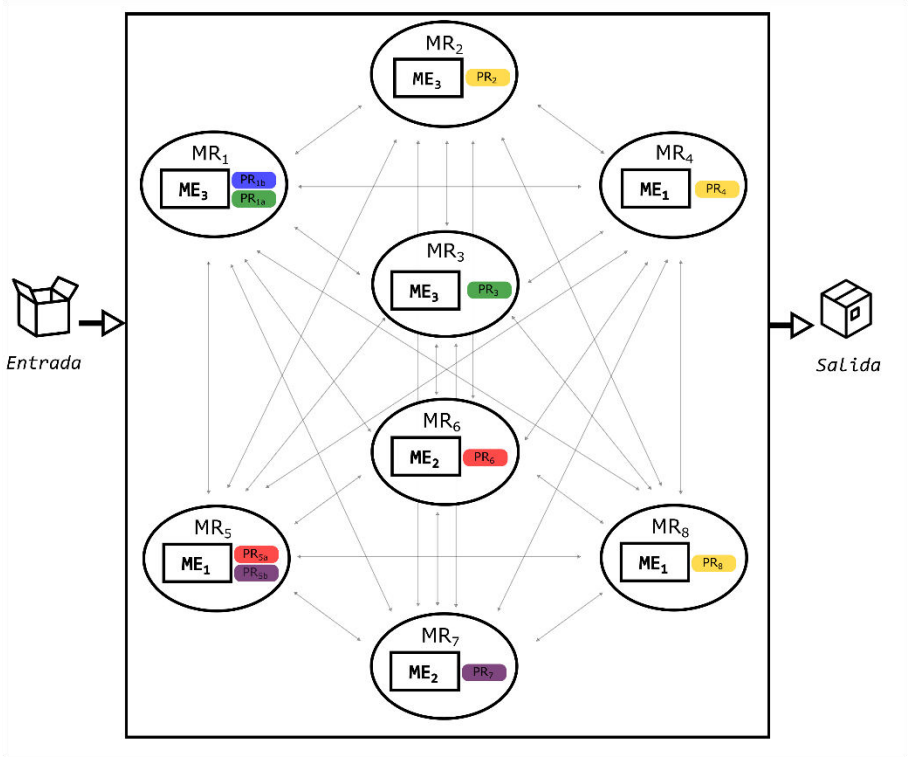

Fig. 2. Modelos RDEVS asociados a la red presentada en la Figura 1.

Sobre la base de este mapeo, en este trabajo se propone una herramienta que facilite la traducción de modelos definidos según la teoría de redes como modelos de red restringidos a modelos de simulación RDEVS codificados en el lenguaje Java.

3 Estructura de la Herramienta Propuesta

La Figura 3 presenta la estructura de la herramienta desarrollada. Como puede observarse, el modelador (haciendo uso de un editor de texto) realiza una especificación textual del modelo de red restringido que da soporte a su escenario de simulación. La gramática libre de contexto RDEVSNL permite abstraer, por medio de su sintaxis, la definición de procesos de enrutamiento en representaciones textuales basadas en nodos y enlaces para describir su estructura. El metamodelo conceptualiza procesos de enrutamiento e incluye un conjunto de restricciones OCL para limitar el modelo de red original, dando lugar a un modelo de red restringido. Luego, la descripción textual definida por el modelador es analizada y validada a partir de la gramática y el metamodelo definidos. Posteriormente, la instancia del modelo validada es traducida a

código Java para obtener su correspondiente representación Java de RDEVS que podrá ser ejecutada en simuladores DEVS.

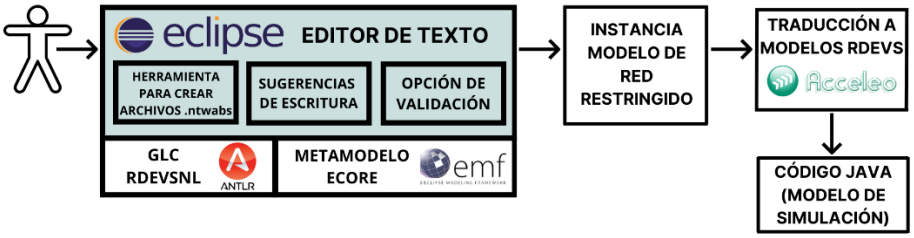

Fig. 3. Esquema del proceso de definición de modelos de simulación a partir del uso del Editor de Texto.

3.1 Gramática Libre de Contexto RDEVSNL

La gramática libre de contexto RDEVSNL, que ha sido presentada en un trabajo previo [12], se basa en un modelo de red restringido para abordar la definición de procesos de enrutamiento a través del formalismo RDEVS, abstrayendo la definición de dichos procesos en representaciones textuales que emplean nodos y enlaces para definir su estructura.

Para su construcción, se generó un analizador léxico que permite reconocer los tokens de una entrada, para así identificar como válidos los símbolos que son escritos en una especificación. A través de él, se definieron las diferentes palabras que son aceptadas por la gramática desarrollada. También, se creó un analizador sintáctico que permite verificar que las palabras y símbolos válidos del lenguaje definido se escriban en estructuras que la gramática permite.

Para mayor versatilidad, se desarrollaron dos versiones de la gramática, una en inglés y otra en español, ambas especificadas e implementadas utilizando ANTLR4. Así, el modelador podrá optar por el idioma de su preferencia para la definición de su modelo.

En la especificación en inglés se pueden identificar tres bloques de construcción primarios: *network, materializes* y *edges*. Es a partir de estos símbolos no terminales que la sintaxis permite definir una red, asignarle un identificador y detallar la lista de nodos que están incluidos en ella, utilizando sentencias del bloque *network*. Esto se puede expresar en múltiples líneas, donde cada nodo es identificado como parte de la red, o detallando la lista completa de nodos (enumerándolos a través de sus identificadores). También, permite establecer el comportamiento interno que ejecutará cada nodo, relacionándolos con la operación de un componente empleando sentencias del bloque *materializes*. Así, un componente define el comportamiento de un nodo o de una lista de ellos. Por último, la sintaxis permite vincular los diferentes nodos (es decir, definir enlaces entre ellos) a través del bloque *edges,* que incluye diferentes sentencias permitidas para definir enlaces entre dos nodos de la red y también hacer uso de listas de múltiples conexiones.

Es importante destacar que la gramática ofrece un conjunto variado de estructuras permitidas para que el modelador pueda optar por utilizar aquellas de su preferencia. La Tabla 2 presenta algunos ejemplos de sentencias válidas para cada bloque. Para visualizar los diagramas de sintaxis de la gramática y obtener mayores detalles, se sugiere revisar [12].

Tabla 2. Ejemplos de expresiones sintácticas permitidas en la versión en inglés de la gramática RDEVSNL.

Bloque de construcción primario	***Ejemplos de sentencias permitidas***
Network	The *RoutingProcess* network includes Machine1, Machine2, *Machine3, Machine4, Machine5, Machine6, Machine7* and *Machine8*
Materializes	The *Machine1, Machine2* and *Machine3* materialize *MACHINETYPEC*
	The behaviors are defined as: (*MACHINETYPEA,Machine8*)
Edges	The output of *Machine3* is connected to the inputs of *Machine4*
	The *Machine4* receives inputs from *Machine6*
	The *Machine1* sends outputs to *Machine2* and *Machine3*

El uso de atributos en el analizador sintáctico de la gramática se asocia con la definición de identificadores tanto para las redes como para nodos y componentes. Los mismos son almacenados como cadenas de caracteres o como arreglos de ellas (cuando se emplean listas de nodos, por ejemplo). Esto es de mucha utilidad para etapas posteriores del proceso de obtención de los modelos RDEVS, ya que a partir de ellos se instanciarán y nombrarán los diferentes elementos del metamodelo sobre el que se basa la gramática. Se darán más detalles de lo mencionado en la Sección 4.

3.2 Metamodelo Ecore (EMF)

Se implementó una versión Ecore del metamodelo que se observa en la Figura 4. Este metamodelo especifica el conjunto de conceptos y relaciones que se mapean con la descripción textual del usuario a fin de instanciar un proceso de enrutamiento válido. Para esto, se utilizó el proyecto EMF de Eclipse [13], que consiste en una herramienta que provee un marco de trabajo para la definición de modelos. El modelo incluye un conjunto de restricciones invariantes definidas en OCL que garantizan la obtención de un proceso de enrutamiento a partir de una definición de red basada en nodos y enlaces.

La Figura 4 presenta el metamodelo que conceptualiza procesos de enrutamiento, donde los estereotipos son utilizados para indicar el componente de red al que refiere el elemento de ruteo. Una especificación RDEVSNL (*NLSpecification*) incluye un proceso de enrutamiento (*Routing Process*). Este concepto refiere al modelo de red (*Network Model*). El modelo de red restringido utilizado para estructurar un proceso de enrutamiento se define sobre un conjunto de nodos (*Node*) que denotan componentes (*Component*). Cada uno de ellos ejecuta el comportamiento de un tipo de compo-

nente (*ComponentType*). A su vez, el proceso de enrutamiento se compone de un conjunto de interacciones dirigidas entre componentes (*Link*), donde uno de ellos actúa como fuente (*source*) y el restante actúa como destino (*destination*). Por otra parte, las referencias resaltadas en color representan las restricciones OCL incluidas en el modelo para garantizar su integridad: *i)* al menos un nodo debe identificarse como inicial, *ii)* al menos un nodo debe identificarse como final, *iii)* los nodos no pueden estar aislados, *iv)* varios enlaces no pueden conectar a los mismos nodos, *v)* los auto-enlaces no están permitidos, *vi)* un componente debe ejecutar un único comportamiento y *vii)* en una especificación RDEVSNL se debe describir a un único proceso de enrutamiento.

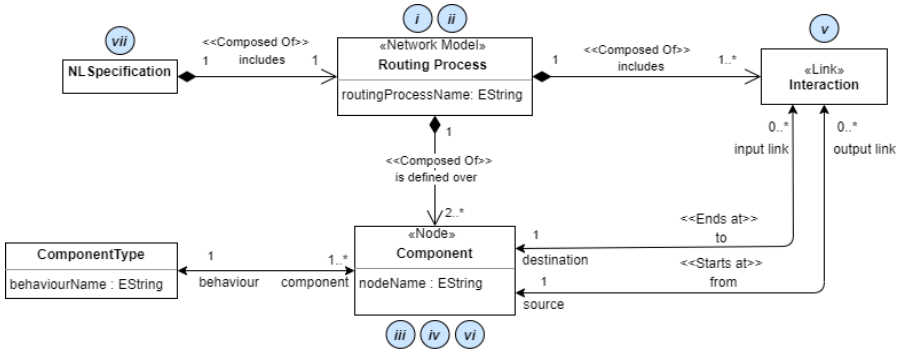

Fig. 4. Metamodelo del dominio instanciado a partir de especificaciones textuales creadas utilizando la sintaxis RDEVSNL.

4 Plugin de Eclipse para la Construcción de Especificaciones en RDEVSNL

Sobre la base de la gramática introducida y el metamodelo presentado en la Sección 3, la herramienta propuesta fue implementada como un plugin para la plataforma Eclipse. Se conforma de un editor de texto que permite instanciar modelos de red restringidos válidos haciendo uso de la sintaxis RDEVSNL, un "wizard" para la creación de archivos con extensiones válidas para el almacenamiento de las descripciones textuales y un proceso de validación que instancia el metamodelo nombrado previamente a fin de verificar la descripción textual. El editor de texto brinda ayudas durante la edición, respetando la correspondencia con el idioma de trabajo seleccionado por el modelador.

El modelador puede seguir los pasos indicados en el "wizard" para la plataforma Eclipse (que fue incluido en el plugin y puede observarse en la Figura 5) para crear un archivo de especificación RDEVSNL, que tendrá extensión ".ntwabs". Puede, dentro de un proyecto Java en Eclipse que actúa como contenedor del archivo a crear, asignarle un nombre y seleccionar el idioma con el que trabajará. Esto último se asocia con el hecho de que la gramática RDEVSNL cuenta con dos versiones: una en inglés y otra en español. Así, esta herramienta soporta la escritura y validación de las especi-

ficaciones en ambos idiomas. Si el modelador no selecciona ningún idioma, por defecto trabajará en inglés.

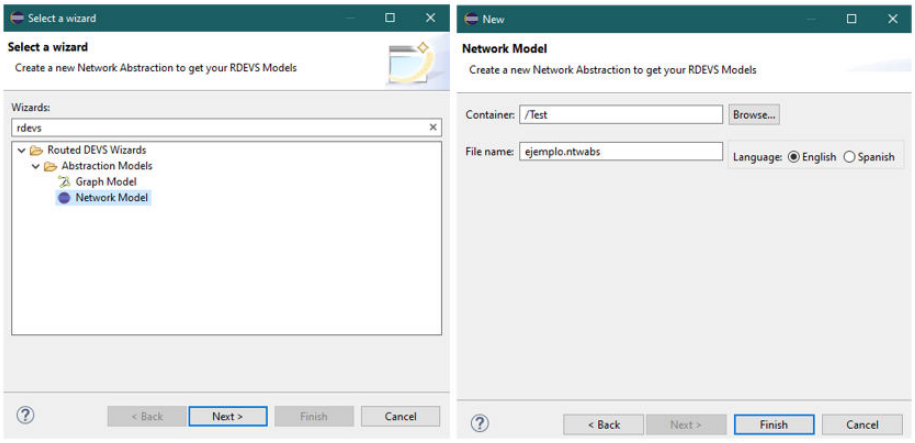

Fig. 5. A la izquierda: primera página del wizard incluido en el plugin, donde se elige la abstracción a utilizar para obtener los modelos RDEVS. A la derecha: segunda página del wizard, donde el modelador puede asignarle un proyecto contenedor y un nombre al archivo, además de elegir el idioma que empleará para su especificación.

4.1 Definición de la Especificación Textual

El editor incluido en el plugin cuenta con ayudas de escritura que facilitan al modelador la edición de sus especificaciones. Entre ellas se encuentra el resaltado de sintaxis. Para lograrlo, las palabras admitidas se clasifican en: *artículos, símbolos no terminales, acciones, comentarios* e *interacciones*. De esta manera, al escribir la especificación, el modelador puede identificar claramente los componentes de cada sentencia a través de sus colores. Por ejemplo, palabras como "*network*", "*node*" o "*component*" (que representan los símbolos no terminales) se resaltan en azul, mientras que palabras como "*includes*", "*sends*" o "*receives*" (que representan acciones) toman el color rojo. Además, puede incluir comentarios cuyo contenido no será analizado durante el proceso de validación de la especificación y se resaltará en color verde. Los mismos pueden ser definidos en una sola línea o en múltiples líneas. En los comentarios de una única línea, deberán incluirse los caracteres "//" al principio. En cambio, si se trata de un comentario de múltiples líneas, se deben incluir los caracteres "/*" al comienzo y "*/" al finalizar.

Con respecto a las sugerencias de escritura, las mismas pueden obtenerse presionando CTRL+SPACE. El modelador podrá seleccionar de a una palabra por vez sobre una lista de palabras sugeridas, haciendo clic en ella o navegando con las flechas de dirección del teclado y presionando ENTER sobre la palabra deseada. Las listas de palabras que se muestran son almacenadas en archivos de configuración, existiendo uno por cada idioma disponible.

El idioma de trabajo elegido por el modelador durante el proceso de creación de su archivo de especificación es almacenado en un archivo de propiedades, que se crea al mismo tiempo que aquel que contendrá la especificación textual. Su extensión es ".properties" y su nombre es el mismo que el elegido para el archivo de especificación. Para lograr esto, se hizo uso de la clase de Java llamada *Properties*, que representa un conjunto persistente de propiedades que son almacenadas en pares *{clave, valor}*. En este caso, la clave es *"language"* y el idioma puede ser *"english"* o *"spanish"*. El hecho de poder recuperar el idioma de trabajo elegido por el modelador permite establecer una correspondencia entre las ayudas de escritura y el idioma con el que se está trabajando, ya que cada vez que se abre el archivo que contiene el texto se recupera del archivo de propiedades (el cual se localiza partiendo de que posee el mismo nombre que el archivo a abrir) el valor de la clave *"language"*, a partir de ciertos métodos que brinda la clase Properties mencionada. Luego, se cargan las ayudas requeridas en consecuencia a dicha elección. Esto también permite conocer con qué versión de la gramática se deberá validar el archivo cuando el modelador decida comenzar el proceso de validación.

Para clarificar la descripción de la herramienta, se presenta un ejemplo. La Figura 6 muestra el proceso de enrutamiento a definir. Este escenario es el mismo que el analizado en la Sección 2 (Figura 1). Como se mencionó previamente, la red está conformada por 8 nodos (representados por máquinas), que se relacionan a través de interacciones dirigidas, y cada uno se corresponde con un tipo de máquina (es decir, ejecuta el comportamiento de un tipo de máquina). Luego, en la misma figura se muestra la definición de su estructura a partir de las sentencias que ofrece la sintaxis RDEVSNL, haciendo uso del editor de texto.

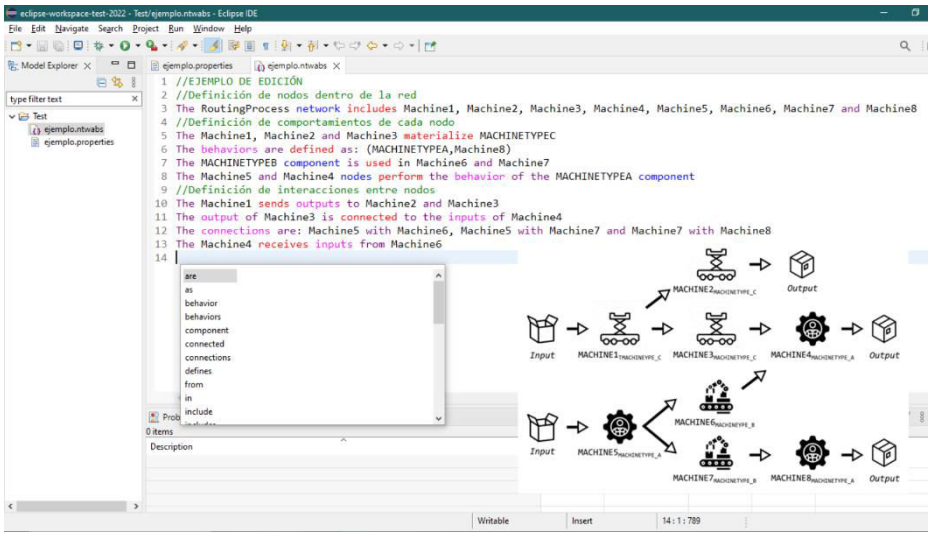

Fig. 6. Captura de pantalla del Editor de Texto incluido en el plugin para admitir la sintaxis. El resaltado de sintaxis y las sugerencias de escritura se hacen visibles, en inglés para este caso. Abajo a la derecha: representación gráfica del proceso de enrutamiento definido en la especificación textual.

La Figura 7 muestra el archivo de propiedades asociado al archivo de especificación que se observa en la Figura 6.

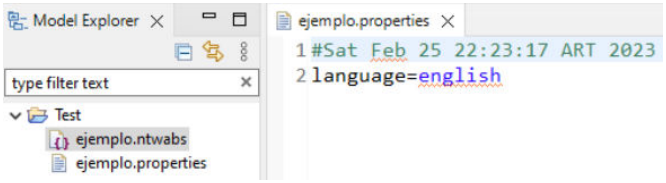

Fig. 7. Captura de pantalla del archivo de propiedades que contiene registro del idioma de trabajo para el ejemplo de la Figura 6.

4.2 Validación de la Especificación Textual

El plugin implementado cuenta con la opción de validar el archivo de especificación, utilizando el analizador sintáctico (o parser) creado con ANTLR4. Cuando el modelador activa el proceso de validación presionando la opción ofrecida en el menú de opciones (*"Check RDEVSNL Specification"*), el análisis sintáctico de RDEVSNL es ejecutado sobre el contenido actual del archivo "*.ntwabs". El mismo fue previamente guardado en forma automática para asegurar que el análisis se realiza sobre su versión más reciente. Luego, el parser trata de reconocer las estructuras de las sentencias a partir de un flujo de tokens, dado por el contenido actual de la especificación.

Si dicho análisis es exitoso (es decir, todas las sentencias que el modelador utilizó para crear su especificación son válidas), utilizando los tokens identificados por el parser se crea automáticamente una instancia del metamodelo Ecore presentado en la Figura 4. ANTLR genera, junto con otras clases Java, una interfaz que contiene los métodos necesarios para recorrer el árbol de sintaxis y manejar eventos como ingresar a una regla de producción o salir de ella. Haciendo uso de dicha interfaz se evalúan las estructuras empleadas y se reconocen los elementos que la componen. Así se identifican las clases del metamodelo que deben instanciarse, junto con los valores a asignar a sus respectivos atributos (utilizando los valores almacenados en cadenas de caracteres y arreglos a través de los atributos de la gramática, mencionados en la Sección 3.1). Cada elemento definido en la especificación es mapeado a un concepto o relación. Cuando una sentencia utilizada incluye una lista de nodos o de conexiones, se utilizan estructuras iterativas para recorrerla y crear todas las instancias referidas en ella, cada una con sus correspondientes atributos.

La Figura 8 muestra una regla de producción perteneciente al bloque de construcción primario *Network.* La misma es utilizada en la línea 3 del ejemplo de la Figura 6 para definir la red y los nodos que la componen, detallándolos en una lista.

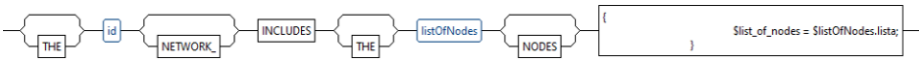

Fig. 8. Regla de producción perteneciente al bloque de construcción primario *Network*.

La Figura 9 presenta la porción del código Java utilizado al salir de una regla del bloque de construcción primario *Network* (que sirve para definir una red, asignarle un

nombre y crear los nodos que la misma incluye). Para eventos asociados a los bloques *Edges* y *Materializes* se utilizan los métodos *void exitEdges(RDEVSParser.EdgesContext ctx)* y *exitMaterializes(RDEVSParser.MaterializesContext ctx),* respectivamente. La clase que los contiene extiende de una clase generada por ANTLR, la cual cuenta con implementaciones vacías de los métodos que reconocen los eventos de entrada y salida a las reglas. Así, los métodos mencionados son implementaciones particulares de los métodos de la clase padre.

```
@Override
public void exitNetwork(RDEVSParser.NetworkContext ctx) {
	Optional<RoutingProcess> routingProcess = this.incluyeRP(ctx.g.getText());
	RoutingProcess newRoutingProcess;
	//Se verifica que el RoutingProcess no haya sido creado por otra sentencia previamente
	if(routingProcess.isEmpty()) {
		//Si no fue creado previamente, se genera una instancia de RoutingProcess
		newRoutingProcess = (RoutingProcess) createInstanceFromModel("RoutingProcess");
		//A la instancia creada se le asigna el nombre
		newRoutingProcess.setRoutingProcessName(ctx.g.getText());
		newRoutingProcess.setNlspecification(this.specification);}
	else {
		//Ya fue creada una instancia de RoutingProcess
		//Se verifica que la misma tenga asignado un nombre
		if(routingProcess.get().getRoutingProcessName()==null) {
			routingProcess.get().setRoutingProcessName(ctx.g.getText());}
	newRoutingProcess = routingProcess.get();}
	//Se verifica el tipo de regla del bloque Network
	if(ctx.list_of_nodes.size()>0) {
		//Es una regla que incluye una lista de nodos
		for(int i=0;i<ctx.list_of_nodes.size();i++) {
			String node = ctx.list_of_nodes.get(i).toString();
			Optional<Component> existing_component = this.incluye(node);
			if(existing_component.isEmpty()) {
				Component newNode = this.createComponent(node);
				newNode.setRoutingprocess(newRoutingProcess);
		}}}
	else if(!(ctx.i == null)) {
		//Es una regla que sólo incluye un nodo
		String node = ctx.i.getText();
		Optional<Component> existing_component = this.incluye(node);
		if(existing_component.isEmpty()) {
			Component newNode = this.createComponent(node);
			newNode.setRoutingprocess(newRoutingProcess);}}
}
```

Fig. 9. Código Java del método *void exitNetwork(RDEVSParser.NetworkContext)*.

El resultado de la ejecución del método *exitNetwork(RDEVSParser.NetworkContext)* es la creación de una instancia de *RoutingProcess* (si el mismo no fue creado previamente por el tratamiento de otra sentencia que causó su creación), y/o la creación de una/s instancia/s de *Component*. Que se genere una o varias instancias depende de si se empleó una regla que incluye un nodo individual o una lista de nodos, además de si los elementos mencionados en la sentencia no fueron creados previamente (por la misma razón mencionada respecto a la instancia de *RoutingProcess*). Del contexto se recuperan los valores almacenados en los atributos de la gramática (como *id* y *listOfNodes* de la Figura 8), para así asignar los identificadores a las diferentes instancias creadas. Existen ciertas reglas de producción del bloque *Network* cuyo uso no causaría la creación de instancias de la clase *Component*, porque sólo buscan asignar un nombre a la red y no mencionan los nodos que son parte de ella (por ejemplo *"the network is called RoutingProcess"*). En el tratamiento de una sentencia que cumple con la estructura de la regla presentada en la Figura 8, se generaría una instancia de RoutingProcess (si no existía previamente) y se emplearía

la estructura repetitiva codificada para crear los diferentes nodos de la lista descripta (que, de igual manera, no existían antes) y darles valores a sus identificadores.

Para el ejemplo de la Figura 6, a partir del contenido de la línea 3 se creará una instancia del concepto *NL Specification* que contendrá una instancia de *RoutingProcess* llamada *"RoutingProcess"*, ocho instancias de *Component* que guardan en el atributo *nodeName* los valores *"Machine1"* hasta *"Machine8"* y las relaciones *IsDefinedOver* para vincular a cada *Component* con *RoutingProcess*. Luego, se crean las relaciones de cada *Component* con las tres instancias de *ComponentType* (a las cuales se les asignaron los valores *"MACHINETYPEA"*, *"MACHINETYPEB"* y *"MACHINETYPEC"* en el atributo *behaviourName*) analizando las líneas 5 a 8. Por último, se mapean los diferentes enlaces (*Interaction*) que se establecen entre los *Component* en las líneas 10 a 13. A cada relación se le especifica el componente que actúa como origen y el que actúa como destino. La Figura 10 muestra las diferentes instancias de las clases del metamodelo del dominio que fueron creadas al analizar la especificación textual del ejemplo en análisis.

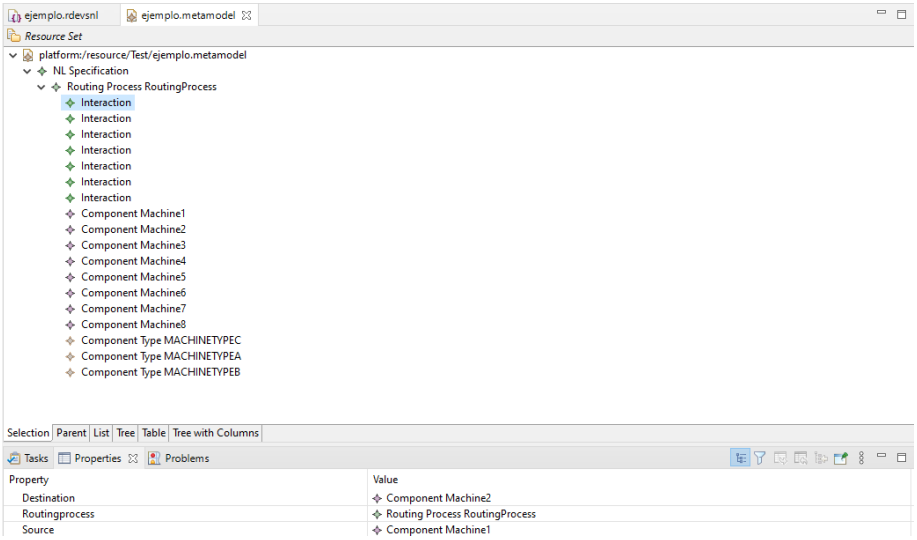

Fig. 10. Instancia del metamodelo Ecore presentado en la Figura 4, obtenido de la validación de la especificación del ejemplo presentado en la Figura 6.

Por otro lado, si se identifican errores sintácticos, se mostrará un mensaje de error. La Figura 11 muestra la Vista *Problemas* de Eclipse, donde se detallan los errores sintácticos identificados para su localización, permitiendo su posterior corrección.

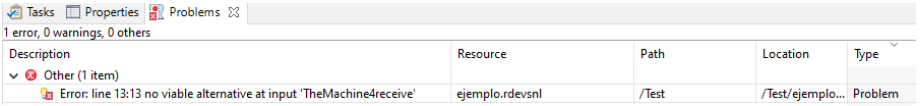

Fig. 11. Vista *Problemas* de Eclipse informando el error sintáctico asociado a la línea 13 de la especificación de la Figura 6, que se origina al reemplazar la palabra *"receives"* por *"receive"*.

Teniendo ya creada la instancia del metamodelo a partir de la descripción textual, el plugin ejecuta su validación para asegurar su correctitud. Aquí se verifican los conceptos, relaciones, multiplicidades y restricciones de OCL del metamodelo sobre la instancia obtenida. Si no se encuentran errores, el modelador recibirá un mensaje de éxito. De lo contrario, podrá visualizar el error identificado (como en la Figura 11). Contando con la posibilidad de acceder a las propiedades del error (como muestra la Figura 12), podrá corregir su descripción del proceso de enrutamiento y volver a realizar las comprobaciones.

En este punto es importante destacar que los elementos utilizados del proceso de enrutamiento corresponden a los modelos RDEVS que definen estructura. El comportamiento interno de las máquinas en el ejemplo presentado no se especifica junto con la descripción textual de la estructura del escenario.

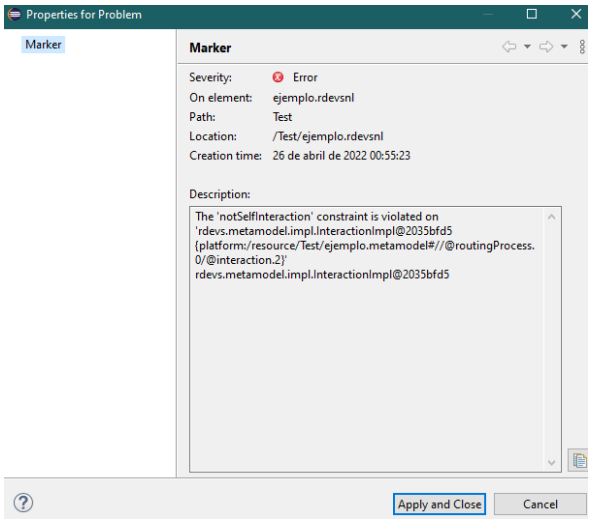

Fig. 12. Propiedades del error que fue mostrado en la Vista *Problemas* de Eclipse, en este caso originado al modificar la línea 11 de la especificación de la Figura 6 por *"The output of Machine3 is connected to the inputs of Machine3"*. Por la restricción *v)* presentada en la Figura 4, esta situación no es admitida.

4.3 Generación de Código Java asociado a los Modelos RDEVS

Para obtener las implementaciones de los modelos RDEVS asociados a una especificación textual previamente validada, se empleó la Librería RDEVS. La misma fue desarrollada como una extensión de DEVSJAVA [14] y provee una solución para implementar modelos de simulación RDEVS en Java. Así, a partir de la instanciación de las clases necesarias que pertenecen a la librería, que se relacionan con los conceptos incluidos en el formalismo, se puede obtener la implementación de modelos de simulación RDEVS ejecutables en el entorno Java.

Para crear las clases que extienden de aquellas pertenecientes a la librería se utilizó Acceleo [15], una tecnología basada en plantillas para crear generadores de código personalizados. Permite producir automáticamente cualquier tipo de código fuente desde un modelo de datos disponible en formato EMF.

La Figura 13 muestra un esquema del proceso de generación automática de código. Para obtener las implementaciones de los modelos RDEVS asociadas a las especificaciones textuales escritas y validadas, a través de Acceleo se navegan los elementos incluidos en la instancia del metamodelo de la Figura 10 para traducirlos a código fuente Java. El código fuente que resulta de este proceso se corresponde con extensiones de las implementaciones de las clases Java pertenecientes a la librería mencionada, para así lograr modelos RDEVS válidos que puedan ser ejecutados posteriormente.

Fig. 13. Proceso de generación de código Java a través de Acceleo.

Si se continúa con el ejemplo presentado previamente, el código Java generado incluiría las clases detalladas en la Tabla 3. Para cada *Component* (desde *Machine1* a *Machine8*) se crea una extensión de la clase *RoutingModel.java* cuyo nombre de clase es obtenido a partir de los atributos que identifican a cada máquina. Las extensiones de la clase *RoutingFunctionElement.java* permiten definir la funcionalidad de ruteo de los modelos de ruteo, teniendo en cuenta los enlaces de dicho componente. Por otro lado, una extensión de la clase *NetworkModel.java* se define a partir del proceso de enrutamiento asociado al escenario especificado. Por último, para cada *ComponentType* se genera una clase que extiende de *EssentialModel.java*, nombrada a partir de los identificadores detallados en la especificación para cada uno de estos elementos.

Tabla 3. Clases Java generadas para el ejemplo de la Figura 6.

Instancia del metamodelo		***Implementación RDEVS***	
Nombre	**Tipo**	**Clase generada**	**Extiende de**
MACHINE-TYPEA	Component Type	ComponentTypeMACHINETYPEA.java	EssentialModel.java
		StateMACHINETYPEA.java	State.java
MACHINE-TYPEB	Component Type	ComponentTypeMACHINETYPEB.java	EssentialModel.java
		StateMACHINETYPEB.java	State.java
MACHINE-TYPEC	Component Type	ComponentTypeMACHINETYPEC.java	EssentialModel.java
		StateMACHINETYPEC.java	State.java
Machine1	Component	NodeMachine1.java	RoutingModel.java
		NodeMachine1RoutingFunctionDefinition.java	RoutingFunction.java

		NodeMachine1RoutingFunctionElement	RoutingFunc-tionElement.java
Machine2	Component	NodeMachine2.java	RoutingModel.java
		NodeMa-chine2RoutingFunctionDefinition.java	RoutingFunction.java
		NodeMachine2RoutingFunctionElement	RoutingFunc-tionElement.java
Machine3	Component	NodeMachine3.java	RoutingModel.java
		NodeMa-chine3RoutingFunctionDefinition.java	RoutingFunction.java
		NodeMachine3RoutingFunctionElement	RoutingFunc-tionElement.java
Machine4	Component	NodeMachine4.java	RoutingModel.java
		NodeMa-chine4RoutingFunctionDefinition.java	RoutingFunction.java
		NodeMachine4RoutingFunctionElement	RoutingFunc-tionElement.java
Machine5	Component	NodeMachine5.java	RoutingModel.java
		NodeMa-chine5RoutingFunctionDefinition.java	RoutingFunction.java
		NodeMachine5RoutingFunctionElement	RoutingFunc-tionElement.java
Machine6	Component	NodeMachine6.java	RoutingModel.java
		NodeMa-chine6RoutingFunctionDefinition.java	RoutingFunction.java
		NodeMachine6RoutingFunctionElement	RoutingFunc-tionElement.java
Machine7	Component	NodeMachine7.java	RoutingModel.java
		NodeMa-chine7RoutingFunctionDefinition.java	RoutingFunction.java
		NodeMachine7RoutingFunctionElement	RoutingFunc-tionElement.java
Machine8	Component	NodeMachine8.java	RoutingModel.java
		NodeMa-chine8RoutingFunctionDefinition.java	RoutingFunction.java
		NodeMachine8RoutingFunctionElement	RoutingFunc-tionElement.java
RoutingProcess	Network Model	NetworkRoutingProcess.java	NetworkModel.java
		InputTranslationFunctionRoutingPro-cess.java	InputTranslationFunc-tion.java

De esta manera, cada tipo de modelo incluido en el formalismo RDEVS (*modelo esencial*, *modelo de ruteo* y *modelo de red*) queda definido a través de una clase específica que es generada. Haciendo referencia a los modelos RDEVS presentados en la Figura 2, los modelos esenciales asociados a cada tipo de máquina se definen a partir de las clases que extienden de *EssentialModel.java*, los modelos de ruteo asociados a cada máquina se definen a través de las clases que extienden de *RoutingModel.java*, y el modelo de red se define con la clase que extiende de *NetworkModel.java*. Además de las mencionadas, otras clases son generadas para dar soporte a la definición de otros elementos relacionados al formalismo.

5 Conclusiones y Trabajos Futuros

Se ha presentado una herramienta de software para M&S implementada como un complemento para el entorno Eclipse. Sus funcionalidades asisten a los modeladores durante el proceso de M&S en RDEVS, ofreciéndoles un entorno de modelado donde pueden definir y validar la estructura de procesos de enrutamiento a través de especificaciones textuales, abstrayéndolos así de las complejidades matemáticas propias del formalismo y aquellas asociadas a la programación de sus modelos.

La traducción a modelos RDEVS desarrollada utilizando Acceleo y la Librería RDEVS permite generar de manera automática el código Java asociado a los modelos de simulación, partiendo de una instancia de un modelo de red obtenida a partir de un proceso de enrutamiento especificado textualmente.

De esta manera, a través de esta herramienta los modeladores podrán obtener, desde la creación de especificaciones textuales de procesos de enrutamiento como redes, modelos de simulación RDEVS sin la necesidad de emplear lenguajes de programación. La gramática RDEVSNL ofrece, a partir de las sentencias permitidas, un lenguaje más cercano al modelador que ayuda a entender con más claridad lo que se está modelando y permite compartirlo fácilmente entre los participantes del proceso de M&S, sin que tengan conocimientos específicos. Contar con un archivo de propiedades como el que se genera al crear un archivo de especificación para almacenar el idioma de trabajo elegido, posibilita a futuro almacenar otros datos de interés que puedan darse en relación con el proceso.

Es importante destacar que las herramientas existentes que ofrecen funcionalidades similares están asociadas al empleo de DEVS y no son directamente aplicables a RDEVS, por tratarse este último de un formalismo reciente. En [16] se ha desarrollado una herramienta para la estructuración y captura de todos los eventos que tienen lugar cuando se ejecuta la simulación de modelos RDEVS. Como trabajo futuro, se buscará integrar la herramienta propuesta en este trabajo con la herramienta de tracking de modelos de simulación RDEVS mencionada. Esto permitirá al modelador seleccionar, en una etapa previa a la obtención de un modelo ejecutable, qué eventos desea capturar en la ejecución de las simulaciones. Para lograrlo, se prevé extender el lenguaje de especificación textual para facilitar al modelador la inclusión o exclusión de los eventos a capturar.

Agradecimientos

El presente trabajo fue realizado bajo la supervisión del Dr. Silvio Gonnet, a quien me gustaría expresar mi agradecimiento por su ayuda y acompañamiento.

Referencias

1. Blas, M., Gonnet, S. y Leone, H. “Routing Structure over Discrete Event System Specification: A DEVS Adaptation to Develop Smart Routing in Simulation Models”, en *Actas de Winter Simulation Conference*, Las Vegas, USA, Diciembre 2017, pp. 774-785.

2. Zeigler, B., Muzy, A. y Kofman, E. *Theory of modeling and Simulation: Discrete Event & Iterative System computational Foundations*. Academic Press, 3era ed., 2018.
3. Blas, M. y Gonnet, S. "Computer-aided design for building multipurpose routing processes in discrete event simulation models", *Engineering Science and Technology, an International Journal*, vol. 24, n.° 1, pp. 22-34, Febrero 2021. https://doi.org/10.1016/j.jestch.2020.12.006.
4. The Eclipse Foundation. Eclipse. Disponible: https://www.eclipse.org/. Accedido por última vez el 3/3/2023.
5. Hopcroft, J.E., Motwani, R. y Ulman, J.D. *Introduction to Automata Theory, Languages and Computation*. 3era ed. Madrid: Pearson, 2007, pp. 143-184.
6. OMG. Meta Object Facility (MOF) Specification, versión 1.4, 2002.
7. ANTLR4 IDE Eclipse Plugin para ANTLR4. ANTLR. Disponible: https://www.antlr.org/tools. Accedido por última vez el 3/3/2023.
8. Espertino, C., "Herramienta de Modelado Textual como Soporte al Formalismo RDEVS", *Concurso de Trabajos Estudiantiles (EST 2022) - JAIIO 51* (Modalidad virtual), Diciembre 2022, vol. 8, n.º 6, pp. 19-28.
9. Alshareef, A., Blas, M.J., Bonaventura, M., Paris, T., Yacoub, A., Zeigler, B.P. "Using DEVS for Full Life Cycle Model-Based System Engineering in Complex Network Design", en: *Nicopolitidis, P., Misra, S., Yang, L.T., Zeigler, B., Ning, Z. (eds) Advances in Computing, Informatics, Networking and Cybersecurity*, Springer International Publishing, Cham, 2022, pp. 215–266. https://doi.org/10.1007/978-3-030-87049-2_8.
10. Newman, M., Barabasi, A.-L. y Watts, D.J. *The Structure and Dynamics of Networks*. Princeton University Press, 2006.
11. Blas, M., Espertino, C. y Gonnet, S. "Modeling Routing Processes through Network Theory: A Grammar to Define RDEVS Simulation Models", en *Anais do III Workshop em Modelagem e Simulação de Sistemas Intensivos em Software*, Joinville, 2021, pp. 10-19. https://doi.org/10.5753/mssis.2021.17255.
12. Espertino, C., Blas, M. y Gonnet, S. "Especificación de Modelos de Simulación RDEVS: Diseño e Implementación de una Gramática Libre de Contexto", en *Actas de 9° Congreso Nacional de Ingeniería Informática/Sistemas de Información CoNaIISI*, 2021. ISBN 978-950-42-0213-4.
13. The Eclipse Foundation: Eclipse Modeling Project. Eclipse Modeling Framework. Disponible: https://www.eclipse.org/modeling/emf/. Accedido por última vez el 6/3/2023.
14. Sarjoughian, H., y Zeigler, B. "DEVSJAVA: Basis for a DEVS-based Collaborative M&S Environment", *Simulation Series*, 1998, vol 30, pp. 29-36.
15. The Eclipse Foundation. Acceleo. Disponible: https://www.eclipse.org/acceleo/. Accedido por última vez el 3/3/2023.
16. Toniolo, M., "Desarrollo de una herramienta de software basada en Java para la captura de eventos en la simulación de modelos RDEVS", *XXIV Concurso de Trabajos Estudiantiles (EST 2021) - JAIIO 50* (Modalidad virtual), Octubre 2021, pp. 32-41.