Developing RDEVS Simulation Models from Textual Specifications

Clarisa Espertino[1], **Maria Julia Blas**[1,2], **Silvio Gonnet**[1,2]

[1] Facultad Regional Santa Fe (FRSF) – Universidad Tecnológica Nacional (UTN)
Lavaisse 610 – Santa Fe – CP 3000 – Argentina

[2]Instituto de Desarrollo y Diseño INGAR – CONICET & UTN
Avellaneda 3657 – Santa Fe – CP 3000 – Argentina

cespertino@frsf.utn.edu.ar,
{mariajuliablas,sgonnet}@santafe-conicet.gov.ar

Abstract. *The Routed DEVS (RDEVS) formalism provides a formalization for routing process simulation. This paper presents the mapping between constrained network models obtained from textual specifications of routing processes and RDEVS simulation models implemented in Java. The proposal is part of a work-in-progress intended to develop M&S software tools for the RDEVS formalism using well-known abstractions to get the computational models attached to such abstraction models through conceptual mapping. Then, modelers can have simulation models without needing to codify any routing implementation. Benefits are i) reduction of implementation times and ii) simulation model correctness regarding the RDEVS formalism.*

1. Introduction

A formalism provides a set of conventions for specifying a class of objects in a precise, unambiguous, and paradigm-free manner. The Discrete Event System Specification (DEVS) is a modular and hierarchical Modeling and Simulation (M&S) formalism based on systems theory that provides a general methodology for the construction of reusable models [Zeigler et al. 2018]. The Routed DEVS formalism (RDEVS) employs the "embedding routing functionality" strategy over DEVS models to provide routing capability from the simulation model conception [Blas et al. 2022]. Although RDEVS is based on DEVS, it is a new formalism that emerges from the M&S community to address new types of problems. Due to its nature, RDEVS simulation models can be executed using DEVS simulators. For RDEVS modeling, new software tools are needed to help practitioners with the specification task. Software tools based on high-level specifications are preferred to promote implementation-independent modeling. One way to do this is by using routing process definitions to structure RDEVS simulation models.

A routing process is a system of interacting components in which the operation of an element (i.e., a routing process component) and the routing of its outputs depend on what is happening throughout the process [Alshareef et al. 2022]. That means interactions between components depend on local information and external data derived from the process structure. As shown later in this paper, for RDEVS models, local information is related to component behavior, and external data is attached to routing functions. Hence, the RDEVS formalism provides a formalization for routing process simulation.

ESPERTINO, Clarisa; BLAS, Maria Julia; GONNET, Silvio.
Developing RDEVS Simulation Models from Textual Specifications.
In: WORKSHOP EM MODELAGEM E SIMULAÇÃO DE SISTEMAS INTENSIVOS EM SOFTWARE (MSSIS), 4. , 2022, Uberlândia/MG. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2022 . p. 41-50.
DOI: https://doi.org/10.5753/mssis.2022.226306.

In [Blas et al. 2021], we have introduced a context-free grammar for defining routing process structures as a particular case of constrained network models. We have provided a set of syntactical elements and a metamodel to specify a valid representation of a routing process structure as a constrained network model. From such a constrained network model, in this paper, we show how the Java implementation of RDEVS simulation models can be mapped to the elements defined in the constrained network metamodel to build an accurate transformation between both levels (i.e., the textual specification as the high-level abstraction and the Java code as the low-level implementation). The goal is *i)* to provide a transformation process between constrained network models and RDEVS simulation models using the Eclipse technology, and *ii)* to add such a transformation process as a new feature of the M&S software tool introduced in [Blas et al. 2021] for the execution of RDEVS models in DEVS simulators as discrete-event simulation models.

The remainder of this paper is organized as follows. Section 2 introduces the background and motivation of our proposal. It also presents the RDEVS library used at the core of the transformation. Section 3 summarizes the textual specification that supports the routing process definition and the software tool attached. Section 4 defines the transformation process from the textual specification to Java classes attached to RDEVS implementations. Section 5 is dedicated to discussing our results and their relation with existing proposals. Finally, Section 6 is devoted to conclusions and future work.

2. Related Work

The RDEVS formalism has been presented as an extension of DEVS in [Blas et al. 2022]. RDEVS is an adaptation of *Classic DEVS* [Zeigler et al. 2018] that adds routing features to the models by introducing a new modeling level: *routing behavior*.

DEVS models are designed to provide *behavior* and *structure* definitions through *atomic* and *coupled* models, respectively. In RDEVS, three types of models are formalized: *essential*, *routing*, and *network* models. These models take advantage of DEVS modeling levels by partitioning the *behavior* into two distinct modeling levels: *domain behavior* and *routing behavior*. The *RDEVS essential model* defines a DEVS atomic model that specifies a *domain behavior* (i.e., the primary behavior of a component). The *routing model* defines a container for an *essential model* that uses a routing policy to manage its inputs and outputs (i.e., it adds a *routing behavior* over a *domain behavior*). Finally, the *network model* defines a set of *routing models* coupled all-to-all to leave the routing functionality to routing policies (i.e., it describes a *structure* over *routing behaviors*). Hence, based on the use of routing policies, RDEVS simulation models provide a formalization of routing processes through the use of three modeling levels: *domain behavior*, *routing behavior*, and *structure*. Each model belongs to a modeling level and plays a role in the routing process specification (see Table 1).

Centered in the M&S framework of DEVS [Zeigler et al. 2018], RDEVS models can be executed using DEVS simulators. Such a framework shows how M&S activities are related by using four conceptual entities [Zeigler and Nutaro 2016]: *i)* the *source system* as the real/virtual environment to be modeled, *ii)* the *model* as the set of instructions for generating data comparable to the data observable in the *system*, *iii)* the *simulator* as the computation specified in the *model*, and *iv)* the *experimental frame* as the conditions under which the *source system* is observed. In this way, it emphasizes the notion

RDEVS Model	Modeling Level	Role in Routing Process
Essential model	Domain behavior	Component behavior
Routing model	Routing behavior	Component routing functionality
Network model	Structure	Process routing functionality

Table 1. RDEVS simulation models, modeling levels, and routing processes. For simplicity, we consider a routing process as a set of linked components.

of *model* and *simulator* as two independent entities linked by the *simulation* relationship when a *model* is executed on a computational environment (i.e., a *simulator*). Since RDEVS formalism is proposed as a subclass of DEVS for modeling new types of problems [Blas et al. 2018], the *model* entity of RDEVS formalism can be seen as a subclass of the DEVS *model* entity.

In conceptual modeling theory, the subtyping used to represent class-subclass dependence reflects the concepts at a more detailed specification level [Parsons and Wand 1997]. Hence, any subclass of the DEVS *model* must be more specific than the DEVS *model*. For RDEVS, introducing a new modeling level is the core of such specification. Given that the RDEVS *model* entity inherits a base set of the superclass properties (i.e., the properties of the DEVS *model* entity), the *simulator* used for its execution could be the same (i.e., the DEVS abstract simulator can execute RDEVS simulation models). However, due to the modeling distinction between DEVS and RDEVS, new modeling strategies are needed to support the development of RDEVS models.

One way to do this is by using abstraction models to design routing processes and then getting the computational models attached to such abstraction models through conceptual mapping. Abstraction creates a conceptual model by extracting only those elements needed for addressing a modeler's concerns. Software tools with such a feature reduce the knowledge required for building discrete-event simulation models (in our case, for routing processes), and modeling tasks can be performed by anyone that understands the problem domain through the abstraction.

Following this approach in [Blas and Gonnet 2021], we abstract the routing process definition into a graph model based on nodes and edges. Instead of using a modeling language, we define a graphical representation for routing processes based on *i)* two types of nodes (one for the component behavior and the other for the component routing functionality) and *ii)* two types of links (one for the routing path and the other for linking the component behavior with its routing functionality). The graph representation is detailed in a metamodel that supports the instantiation of valid abstractions. These abstractions are mapped to the modeling levels presented in Table 1 to define RDEVS equivalences. Then, we show how the metamodel can be deployed using Eclipse Modeling Framework [The Eclipse Foundation: Eclipse Modeling Project 2022]. Furthermore, we present a plug-in for Eclipse [The Eclipse Foundation 2022b] that supports the graphical modeling of routing processes (through metamodel instantiation) as the core specification for the creation of the related RDEVS models. To integrate the implementation of RDEVS models with the Eclipse technology used to support the graph metamodel, the simulation models obtained from the routing process definition were developed as Java classes that extend the RDEVS library (Section 2.1). In this way, modelers can have RDEVS simulation models without needing to program any routing implementation.

Aiming to continue using abstraction models to define routing processes, in [Blas et al. 2021], we have presented a textual representation based on network theory. Such a textual representation (supported by the context-free grammar detailed in Section 3) can be used as an alternative specification of the graphical representation based on graphs. In Section 4, we show how conceptual elements defining the constrained network model attached to a routing process are mapped to the set of Java classes required to implement the related RDEVS simulation models.

2.1. The RDEVS Library (Java Implementation of RDEVS Simulation Models)

As stated before, the RDEVS formalism is a subclass of DEVS. This means that existing implementations of DEVS can be used to support RDEVS implementations. From this perspective, a Java library was developed using DEVSJAVA [Sarjoughian and Zeigler 1998] as the underlying M&S layer. DEVSJAVA is a software tool implemented in Java that supports defining models in DEVS formalism through an object-oriented conceptualization. By extending DEVSJAVA, the RDEVS library provides a solution for implementing RDEVS simulation models in Java and, later, executing these models using the DEVSJAVA engine.

Figure 1 illustrates the main classes of the RDEVS library using a UML class diagram. As the figure shows, all the concepts included in the formalism were defined as Java classes. For example, the library includes a Java class for each type of RDEVS model defined in the formalism (i.e., *EssentialModel.java*, *RoutingModel.java*, and *NetworkModel.java*). Relationships are used to denote dependencies among classes. Then, for example, the library defines the routing policy of a routing model as the *RoutingFunction.java* linked to the *RoutingModel.java* definition as the *delr* property of the related *Omega.java* class (which is linked to the model by the *w* property). Moreover, the *NetworkModel.java* class is related to one instance of *InputTranslationFunction.java* class through the *Tin* property to define the transformation of input events to input events with identification (i.e., instances of the *IdentifiedEvent.java* class). A similar strategy is used to relate the *NetworkModel.java* with the *OutputTranslationFunction.java*. The *IdentifiedEvent.java* class is used to denote the structure of events managed by the models with routing functionality.

The diagram includes a few operations for the classes that represent RDEVS simulation models. These operations are added to illustrate how the library supports the behavioral definition of routing processes. Considering the modeling levels detailed in Table 1, the classes defined in the library support the *structure* definition (i.e., the set of classes allows implementing the *NetworkModel.java* class using accurate routing model definitions). The *routing behavior* level is supported by the set of operations defined as "final" in *RoutingModel.java*. These operations (e.g., *delext()*, *delint()*, and *out()*) are implemented as Java code in the library and cannot be changed in further subclasses. In this way, the library ensures the correctness of the routing functionality defined at the core of the RDEVS formalism. Finally, the operations defined as "abstract" in the *EssentialModel.java* class support the *domain behavior*. As with any abstract element, these operations require a Java implementation in further subclasses to define the behavioral specification of components included in the routing process. Since the behavioral specification of such components is part of the domain problem, the RDEVS library only provides the interface required for their simulation.

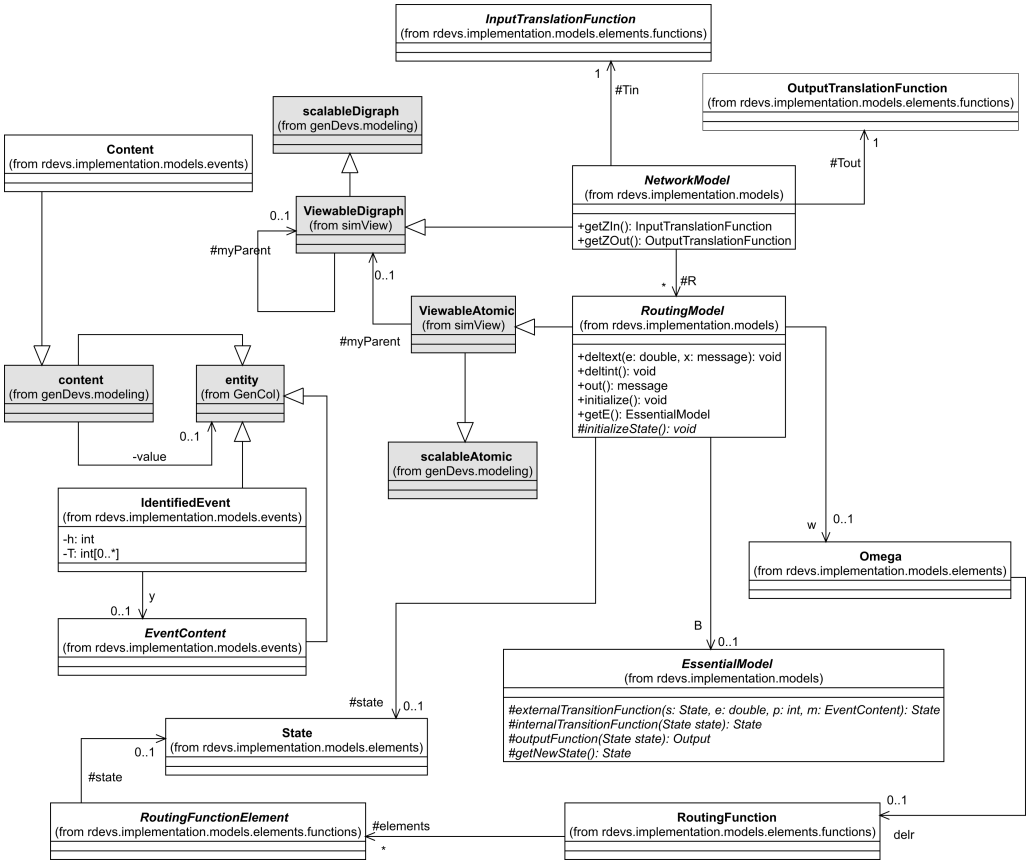

Figure 1. UML class diagram of the Java classes included in the RDEVS library. Classes highlighted in gray belong to the DEVSJAVA package.

The Java classes detailed in the RDEVS library are designed as extension points for building RDEVS implementations. An extension point is the definition of the provided interface for extensions [Klatt and Krogmann 2008]. That is, an extension itself of the classes that represent RDEVS models is an implementation of the RDEVS model according to the extension point defined in the library. Hence, the RDEVS library includes extension points configured for designing explicit instances of RDEVS models as reusable components slated for executing the routing simulation without any other consideration.

Following this consideration, each element included in an abstraction model that defines a routing process can be mapped to an implementable RDEVS model over an extension point of the library (i.e., a new Java class based on the existing ones).

3. Defining Routing Processes using Constrained Network Models

3.1. The Textual Specification: RDEVSNL

The context-free grammar RDEVSNL [Blas et al. 2021] is based on a constrained network model to approach the definition of routing processes through the RDEVS formalism. It abstracts the definition of these processes into textual representations using nodes and links to describe their structure.

Two versions of the syntax were developed, one in English and the other in Spanish. Both were specified and implemented using ANTLR4 [Parr 2022]. In the English

specification, three primary building blocks can be identified: *i) network*, *ii) materializes*, and *iii) edges*. When a RDEVSNL specification is used to structure a routing process, a *Network* is defined to model that process. The sentences from the *network* block allow to assign an *id* and describe the *list of nodes* that are part of the network. This can be done in a unique specification (e.g., the A and B nodes are part of a C network) or in multiple text lines (e.g., the A node is part of C network; the C network includes B node). As well, in a routing process, each *component* exhibits an internal operation and defines the behavior of a node or list of nodes. Sentences from the *materializes* block can be used to establish the behavior that each node will execute, that is associated with the internal operation of a component (e.g., the D component defines the behavior of A and B nodes; the A node performs the behavior of D component). Moreover, links define directed interactions between nodes. The grammar enables the definition of these interactions in multiple ways, using sentences from the *edges* block (e.g., the A node sends outputs to B node; the B and E nodes receive inputs from the A node). Table 2 summarizes some of the syntactical expressions included in RDEVSNL.

To instantiate valid routing processes from textual specifications, the metamodel illustrated in Figure 2 was developed using EMF [The Eclipse Foundation: Eclipse Modeling Project 2022]. Stereotypes are used to indicate the network model component to which the routing element refers. A RDEVSNL specification (*NLSpecification*) includes a *Routing Process*. This concept is associated with the *Network Model*. The constrained network model used to structure a routing process is defined over a set of nodes (*Node*), where each of them denotes a *Component* and executes the behavior of a *Component Type*. Moreover, the routing process is composed of a set of directed interactions between components (*Link*), where one of them acts as a *source* and the other acts as a *destination*.

In Figure 2, the notes highlighted in gray detail OCL constraints added to ensure obtaining a routing process from a network model definition. This is what we called *the constrained network model*. These constraints guarantee the integrity of the metamodel as follows: *i)* at least one node must be identified as initial, *ii)* at least one node must be identified as final, *iii)* nodes cannot be isolated, *iv)* multiple links cannot connect the same pair of nodes, *v)* self-links are not allowed, *vi)* a component must execute a single behavior, and *vii)* a single routing process must be described in a RDEVSNL specification.

Primary Building Block	Allowed Sentences
Network	The A node is part of C network The C network includes B node The A and B nodes are part of a C network
Materializes	The A node performs the behaviour of D component The B node materializes D component The D component defines the behaviour of A and B nodes
Edges	The A node sends outputs to B node The E and F nodes receive inputs from the B node The connections are: A with B, B with E and B with F

Table 2. Examples of syntactical expressions allowed in the context-free grammar RDEVSNL (English version).

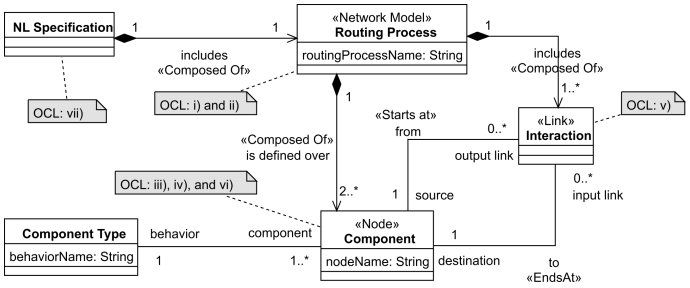

Figure 2. Metamodel used to validate the routing process definition from a constrained network model specification.

3.2. The Plug-in for Eclipse: RDEVSNL Editor

Based on the Ecore metamodel, a new plugin for the Eclipse platform was developed. This software tool is composed of a text editor that allows specifying valid constrained network models using the RDEVSNL syntax, a "wizard" for creating files with the "*.rdevsnl" extension that stores textual descriptions, and a validation process that instantiates the defined metamodel following the parsing of the textual specification.

The text editor provides writing aids during the edition (such as syntax highlighting and typing suggestions) using the language selected by the modeler. When the validation option for a specification is activated, the RDEVSNL syntax analysis is executed over the current content of the "*.rdevsnl" file. Then, the parser tries to recognize the sentence's structures from a stream of tokens given by the actual specification. If the analysis is successful (i.e., all the sentences that the modeler used to create the specification are valid), using the tokens identified by the parser, an instance of the Ecore metamodel shown in Figure 2 is automatically generated. Afterwards, the metamodel's concepts, relationships, multiplicities and OCL restrictions are verified over the obtained instance. If no issues are found, the modeler will receive a success message that ensures the textual definition of the network model is in correspondence with a valid routing process. On the contrary, the modeler will visualize an error message and a list with more details in the Problems view of Eclipse (having the possibility to fix its specification and check the new content). A video of the tool operation can be seen here.

4. From Textual Specifications to Java Classes

The routing elements used as stereotypes in Figure 2 are equivalent to the RDEVS modeling levels presented in Table 1. These equivalences are used to get the implementation of the RDEVS models attached to the textual specification already validated by the plug-in as an instance of the metamodel. To do this, we use Acceleo [The Eclipse Foundation 2022a].

Acceleo is a template-based technology that allows the creation of code generators from any data source available in EMF format. By defining a generation model for the text-to-code transformation, the elements defined in the abstraction model (i.e., the instance of the metamodel) are navigated to "write" the corresponding Java classes. These classes are created as extensions of the ones defined in the RDEVS library.

Each *Component* included in the *Routing Process* is used to define an extension of the *RoutingModel.java* class. These extensions are named using the value of the *node-*

Name attribute. The routing functionality of the routing models is defined through a new set of extensions of the *RoutingFunctionElement.java* class. These extensions use the *output links* to define available destinations. Accurate sources are defined considering the *input links* as part of the *nodeName* (class) definition. In this way, *Interactions* and *Components* are used to define the *routing behavior* (i.e., the intermediate modeling level detailed in Table 1).

Using the *Routing Process* definition, an extension of the *NetworkModel.java* class is defined. This class refers to the *structure* (i.e., the final modeling level of Table 1). The name of such an extension is defined using the value of the *routingProcessName* attribute. The extension includes the full coupling of all routing model classes that define *Components* of the *Routing Process*.

It is important to denote that the *domain behavior* (i.e., the first modeling level of Table 1) cannot be fully defined from the routing process definition. Each *ComponentType* defined as part of the *NetworkModel* is used only to structure an extension of the *EssentialModel.java* class. Following other extensions, the value of the *behaviorName* attribute is used to name the new subclass. However, the specification of the abstract operations to be redefined at the subclass level cannot be obtained from the *Routing Process* definition. Such a behavioral specification is part of the domain problem. Then, the modeler should detail these operations using Java code or another type of DEVS modeling specification. Since essential models are defined as DEVS atomic models, DEVS modeling tools can be used to achieve the domain behavior specification.

5. Discussion

To illustrate proof of concepts, we follow the example presented in [Blas et al. 2021]. Table 3 summarizes the Java classes obtained during the transformation process executed over the metamodel instances of such an example.

Metamodel Instance		RDEVS Java Implementation	
Name	Type	New class	"extends"
MACHINE_TYPEA	Component Type	MACHINE_TYPEAEssentialModel.java MACHINE_TYPEAState.java	EssentialModel.java State.java
MACHINE_TYPEB	Component Type	MACHINE_TYPEBEssentialModel.java MACHINE_TYPEBState.java	EssentialModel.java State.java
Machine_1	Component	Machine_1RoutingModel.java Machine_1RoutingFunction.java Machine_1RoutingFunctionElement.java	RoutingModel.java RoutingFunction.java RoutingFunctionElement.java
Machine_2	Component	Machine_2RoutingModel.java Machine_2RoutingFunction.java Machine_2RoutingFunctionElement.java	RoutingModel.java RoutingFunction.java RoutingFunctionElement.java
Machine_3	Component	Machine_3RoutingModel.java Machine_3RoutingFunction.java Machine_3RoutingFunctionElement.java	RoutingModel.java RoutingFunction.java RoutingFunctionElement.java
Machine_4	Component	Machine_4RoutingModel.java Machine_4RoutingFunction.java Machine_4RoutingFunctionElement.java	RoutingModel.java RoutingFunction.java RoutingFunctionElement.java
Machine_5	Component	Machine_5RoutingModel.java Machine_5RoutingFunction.java Machine_5RoutingFunctionElement.java	RoutingModel.java RoutingFunction.java RoutingFunctionElement.java
Routing_Process	Network Model	Routing_ProcessNetworkModel.java Routing_ProcessInputTranslationFunction.java Routing_ProcessOutputTranslationFunction.java	NetworkModel.java InputTranslationFunction.java OutputTranslationFunction.java

Table 3. List of the Java classes obtained for the proof of concepts.

The main advantages of our proposal are *i)* reduction of implementation times through fast modeling solutions and *ii)* simulation model correctness regarding the formalism through well-defined and standardized simulation models. We are now concluding the testing of the transformation process.

The plug-in in development will be more suitable than other software tools because it employs a general abstraction model (i.e., a network model) as the core definition of the RDEVS simulation models. This provides a more accurate representation of the problem to the modeler. That is the main difference with other approaches like DEVS Modeling Language (DEVSML) and DEVS Natural Language (DEVSNL).

DEVSML [Mittal and Douglas 2012] provides a platform-independent way to specify DEVS models that are transformed to platform-specific language implementation in Java, C++, or any other programming language. On the other hand, DEVSNL [Zeigler and Sarjoughian 2017] provides a natural language specification to understand FDDEVS (Finite Deterministic DEVS) models. These models can be used to automatically generate DEVS atomic models in Java that have full capability to express messages and states. In both cases (i.e., DEVSML and DEVSNL), the modeler needs to understand how DEVS models are structured to build specifications. Instead, in our case, the modeler is abstracted from the notions of RDEVS formalism and generates an (abstract) network model to represent a problem. Such an abstract model is used to create the related simulation models in Java. The separation of concerns between the abstraction model and the programming language used to support the simulation model implementations allows a further mapping to other programming languages.

6. Conclusions and Future Work

The RDEVS formalism provides a formal definition for the M&S of general routing processes employing the "embedding routing functionality" strategy over DEVS models. In this paper, we have presented a plug-in in development intended to obtain Java implementations of RDEVS models from an abstract model defined in a textual specification. For the textual representation, we propose a context-free grammar based on a constrained network model. Such grammar has been implemented using ANTLR4. A metamodel is used to map the textual definition with valid routing processes. This metamodel allows a direct mapping between its concepts and RDEVS simulation models. Then, Java classes are derived using Acceleo. The RDEVS library is used as support since it enhances the development of RDEVS models in Java using some features provided by DEVSJAVA.

Our proposal is part of a work-in-progress intended to develop M&S software tools for the RDEVS formalism as a discrete-event specification for routing processes. Our final aim is to provide a tool that allows modelers to *i)* define the problem domain using well-known abstractions and *ii)* get the computational models attached to such abstraction models through conceptual mapping. Then, modelers will be able to have simulation models without needing to codify any routing implementation. Moreover, they can get simulation models without having programming skills. The plug-in presented in this paper is a fundamental part of such research as an additional feature of the graphical specifications already defined in [Blas and Gonnet 2021]. Future work is devoted to the development of new representations for large-scale routing processes.

References

Alshareef, A., Blas, M. J., Bonaventura, M., Paris, T., Yacoub, A., and Zeigler, B. P. (2022). *Using DEVS for Full Life Cycle Model-Based System Engineering in Complex Network Design*, pages 215–266. Springer International Publishing, Cham.

Blas, M., Espertino, C., and Gonnet, S. (2021). Modeling routing processes through network theory: A grammar to define rdevs simulation models. In *Anais do III Workshop em Modelagem e Simulação de Sistemas Intensivos em Software*, pages 10–19, Porto Alegre, RS, Brasil. SBC.

Blas, M. J. and Gonnet, S. (2021). Computer-aided design for building multipurpose routing processes in discrete event simulation models. *Engineering Science and Technology, an International Journal*, 24(1):22–34.

Blas, M. J., Gonnet, S. M., Leone, H. P., and Zeigler, B. P. (2018). A conceptual framework to classify the extensions of devs formalism as variants and subclasses. In *2018 Winter Simulation Conference (WSC)*, pages 560–571. IEEE.

Blas, M. J., Leone, H., and Gonnet, S. (2022). Devs-based formalism for the modeling of routing processes. *Softw. Syst. Model.*, 21(3):1179–1208.

Klatt, B. and Krogmann, K. (2008). Software extension mechanisms. *Fakultt fr Informatik, Karlsruhe, Germany, Interner Bericht*, 8:2008.

Mittal, S. and Douglas, S. A. (2012). Devsml 2.0: The language and the stack. Technical report, AIR FORCE RESEARCH LAB WRIGHT-PATTERSON AFB OH.

Parr, T. (2022). Antlr. `https://www.antlr.org/`.

Parsons, J. and Wand, Y. (1997). Choosing classes in conceptual modeling. *Communications of the ACM*, 40(6):63–69.

Sarjoughian, H. S. and Zeigler, B. (1998). Devsjava: Basis for a devs-based collaborative m&s environment. *Simulation Series*, 30:29–36.

The Eclipse Foundation, . (2022a). Acceleo. `https://www.eclipse.org/acceleo/`.

The Eclipse Foundation, . (2022b). Eclipse. `https://www.eclipse.org/`.

The Eclipse Foundation: Eclipse Modeling Project, . (2022). Eclipse modeling framework. `https://www.eclipse.org/modeling/emf/`.

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

Zeigler, B. P. and Nutaro, J. J. (2016). Towards a framework for more robust validation and verification of simulation models for systems of systems. *The Journal of Defense Modeling and Simulation*, 13(1):3–16.

Zeigler, B. P. and Sarjoughian, H. S. (2017). *DEVS Natural Language Models and Elaborations*, pages 43–69. Springer International Publishing, Cham.