Modeling Routing Processes through Network Theory: A Grammar to Define RDEVS Simulation Models

María J. Blas[1,2], Clarisa Espertino[2], Silvio Gonnet[1,2]

[1]Instituto de Desarrollo y Diseño INGAR – Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) – Universidad Tecnológica Nacional (UTN)
Avellaneda 3657 – Santa Fe – CP 3000 – Argentina

[2]Departamento de Ingeniería en Sistemas de Información – Universidad Tecnológica Nacional (UTN) – Facultad Regional Santa Fe
Lavaisse 610 – Santa Fe – CP 3000 – Argentina

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

Abstract. *The Routed DEVS (RDEVS) formalism provides a reasonable formalization for the simulation of routing processes. In this paper, we introduce a context-free grammar for the definition of routing processes as a particular case of a constrained network model. Such grammar is based on a metamodel that defines the semantics over the syntactical elements. This metamodel allows a direct mapping between its concepts and RDEVS simulation models. A Java implementation is provided for the grammar as a plug-in for Eclipse IDE. The main benefit of this software tool is the feasibility of getting a simulation model without having programming skills.*

1. Introduction

Network theory is a useful technique to model relationships between entities [Newman, Barabasi and Watts 2006]. A network consists of nodes connected by a set of links. Such a representation has been widely adopted for modeling studies in several fields, such as, for example, the social network domain [Borgatti and Halgin 2011]. In Software Engineering, studies have used this technique to evaluate systems/software issues [Wen, Kirk and Dromey 2007; Pan 2011; Zakari, Lee and Chong 2018].

When in a system of interacting components, the operation of a component and the routing of its outputs depend on what is happening throughout the process, the dynamic of the system can be seen as a routing process. Routing processes exhibit a strong interdependence within components that allow modeling their structure using network theory. Moreover, when routing processes are studied as discrete-event systems, the Discrete Event System Specification (DEVS) formalism [Zeigler, Muzy and Kofman 2018] provides a solid basis for their Modeling and Simulation (M&S). From this perspective, routing mechanisms can be structured over DEVS models to define routing processes. Such mechanisms can be interpreted as routing functionalities attached to the system behavior. This is the case of the Routed DEVS (RDEVS) formalism [Blas, Gonnet and Leone 2017].

In this paper, we discuss how the RDEVS formalism provides a reasonable formalization for the M&S of routing processes based on a constrained network model.

Moreover, from the experience described in [Blas and Gonnet 2021] where RDEVS simulation models are derived from graphical representations, we propose a natural language syntax for the routing process definition using a constrained network model grammar that allows a further derivation of RDEVS simulation models. Our aim is *i)* to offer a software environment for the natural language definition of routing situations using a constrained network model, and *ii)* to be able to get a model definition that supports a further generation of Java code for such routing situations in a way that they can be executed in DEVS simulators as discrete-event simulation models.

The remainder of this paper is organized as follows. Section 2 introduces the constrained network model that supports the definition of routing processes. Section 3 presents the RDEVS formalism as the M&S core used to define routing processes as simulation models. Section 4 presents the grammar and its implementation for Eclipse Integrated Development Environment (IDE) [The Eclipse Foundation 2021a]. Finally, Section 5 is devoted to conclusions and future work.

2. Routing Processes as Constrained Network Models

A routing process can be defined as *"a system of interacting components in which the operation of a component and the routing of its outputs depend on what is happening throughout the process"*. Here, interactions between components depend on both local and external data. Local data refers to internal information of components, while external data refers to information derived from the process structure.

The definition of a routing process may contain different types of components. Each type of component operates independently. This means that the internal operation of components is defined independently from the structure of the routing process. Since routing depends equally on the operative description of the component and process structure, the component can decide the output destinations. Hence, components can take decisions about routing such as *i)* alternate the routing of its outputs to avoid congestion, *ii)* block the routing of its outputs from entering to a precise sector of predefined components, and *iii)* accelerate/decelerate the processing of its inputs (to produce faster/slower outputs) when knowing that downstream nodes are free/busy.

A conceptualization of routing processes can be defined using constrained network models. Network Theory proposes modeling a system as a set of nodes that are connected by links [Newman, Barabasi and Watts 2006]. Both elements (i.e., nodes and links) can have different meanings. For example, if the network is modeling the roadmap of Argentina, the network defines Argentina as a set of nodes that represents cities along with a set of links (among these nodes) that depicts roads between them. When modeling a routing process as a network, nodes define components, and links denote interactions between them. However, the network model that depicts a routing process should be constrained to ensure model correctness (e.g., components cannot be isolated, self-interactions are not allowed, etc.). These kinds of constraints restrict the original network model giving a constrained network model for representing routing processes. From this perspective, a routing process can be seen as a constrained instance of a network model.

A metamodel is a model which defines the used language to design a model [OMG 2002]. That is, it is a model in which instances are models. It defines (at the metamodeling level) the description of all the concepts, their semantics, and the syntactic rules of a language that is instantiated at the modeling level. Hence, both modeling concepts and relationships are equally defined as metamodel concepts. Metamodels are powerful modeling tools to ensure the correctness of models' structure. Furthermore, they allow validating the model instantiation regarding the rules defined at the metamodeling level. Figure 1(a) presents a Unified Modeling Language (UML) diagram that depicts a metamodel containing the elements to be instantiated for the definition of a concrete network type. In such a metamodel, a *Network Model* is defined as a *Composition Of Network Elements*. Such elements are defined as *Nodes* and *Links*. In this case, *Links* are unidirectional. Hence, each *Link Starts At* a *Node* and *Ends At* a *Node* (possibly the same that the one placed at the beginning).

From the metamodel depicted in Figure 1(a), different types of networks can be instantiated. Figure 1(b) presents an instance of such a metamodel as a UML diagram that describes a routing process as a constrained network model. Stereotypes are used to refer the metamodel elements (i.e., concept or relationship) instantiated at the modeling level. The diagram is restricted with Object Constraint Language (OCL) constraints to ensure the routing process definition. These constraints are detailed in Table 1.

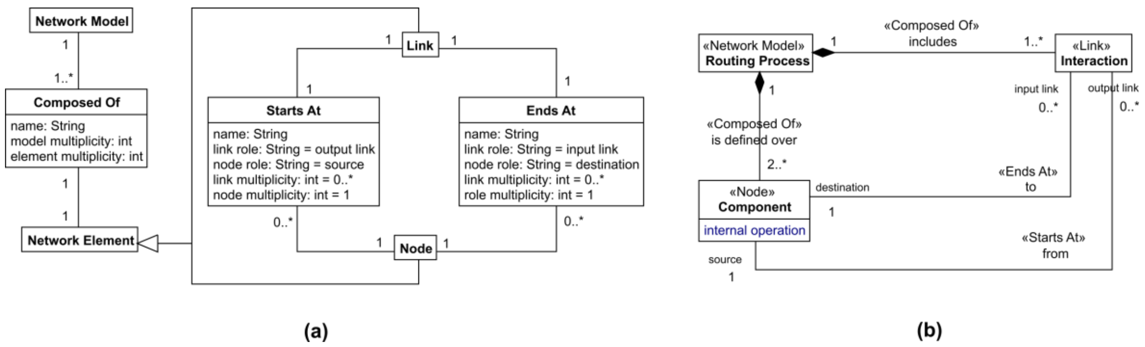


Figure 1. UML class diagrams that support the definition of routing processes as constrained network models. (a) Metamodel that defines the main concepts and relationships required for instantiating a network model. (b) Constrained network model instantiated from the metamodel to represent a routing process.

Table 1. OCL constraints attached to the UML diagram of Figure 1(b).

ID	OCL constraint
C1	`context RoutingProcess` `invariant existsStartingComponent: self.component-> select(c|c.inputLink->size()=0 and c.outputLink->size()>0)->size()>0`
C2	`context RoutingProcess` `invariant existsEndingComponent: self.component-> select(c|c.inputLink->size()>0 and c.outputLink->size()=0)->size()>0`
C3	`context Component` `invariant notIsolated:(self.inputLink->size() + self.outputLink->size()) > 0`
C4	`context Component` `invariant multipleInteractions: self.outputLink->forAll(e1,e2|e1<>e2 implies e1.destination <> e2.destination)`
C5	`context Interaction` `invariant notSelfInteraction: self.source<>self.destination`

When a constrained *Network Model* is used to structure a *Routing Process*, the model *is defined over* a set of *Components*. It also *includes* a set of *Interactions*. At the modeling level, the *Nodes* denote *Components*, and *Links* define directed *Interactions* between *Components*. At least one *Component* should be identified as the starting *Component* (constraint C1). Also, at least one *Component* should be detailed as the ending *Component* (constraint C2). *Components* cannot be isolated (constraint C3). Two *Components* are linked in an *Interaction*. One *Component* acts as the *source* from which the *Interaction* takes place and the other acts as the *destination* to where the *Interaction* is directed. Various *Interactions* cannot connect the same *Components* (constraint C4). Moreover, self-interactions are not allowed (constraint C5).

In a *Routing Process*, each *Component* exhibits an *internal operation*. Such an operation cannot be mapped to the metamodeling level because it is part of the domain description (i.e., the operation is specified when a concrete routing process is defined). However, we introduce this attribute as part of the *Component* concept to provide a full definition of routing process' elements. In the next section, we show how a routing process description can be mapped to discrete-event simulation models.

3. The Routed DEVS (RDEVS) Formalism

The RDEVS formalism employs the "embedding routing functionality" strategy over DEVS models to provide routing capability from the simulation model conception. Such a DEVS extension was presented in [Blas, Gonnet and Leone 2017] as a subclass of the classic DEVS that adds routing features to the atomic model capabilities by introducing a new model: the *routing model*. Hence, the formalism act as a "layer" above DEVS that provides routing functionality without requiring the user to "dip down" to DEVS itself for any functions. To accomplish that, RDEVS defines three models: *essential* (equal to the atomic model), *routing* (new model introduced by the extension), and *network* (like the coupled model). These models are related as follows:

- the *essential model* is formally defined as a DEVS atomic model that specifies a domain behavior,
- the *routing model* is formally defined as a container for an *essential model* that uses a *routing policy* to manage its inputs and outputs (i.e., domain and routing behaviors together), and
- the *network model* is formally defined as a set of *routing models* coupled all-to-all to leave the routing functionality to *routing policies* (i.e., the model structure).

The core of RDEVS is to abstract the event flow into discrete-event models that arrange events independently from the domain behavior of components. The *routing policy* is isolated from the domain behavior allowing the reuse of the *essential model* in distinct *routing models*. Furthermore, the same *network model* can support different configurations by only changing the *routing policies* attached to its *routing models*.

3.1. RDEVS as Formalization of Routing Processes

Formalization makes it easier to work out the implications of an abstraction and implement them in reality [Zeigler, Muzy and Kofman 2018]. As detailed in [Blas and Gonnet 2021], the RDEVS formalism is designed to level out the modeling effort of routing processes modeled in DEVS. Hence, the RDEVS formalism can be used as formalization language for the constrained network model depicted in Figure 1(b).

Each type of RDEVS model can be used to formalize an element defined in a routing process as follows: *i)* the *network model* structures the *Routing Process* (i.e., the *Network Model* at metamodeling level) as a composition of *routing models*; *ii)* the *routing model* represents a *Component* (i.e., a *Node* at metamodeling level) included in the *Routing Process*; and *iii)* the *essential model* describes the *internal operation* of a *Component*. For the formalization of a *Component*, the *routing model* is defined as the pair *{ routing policy, essential model }* that verifies the *routing policy* over the incoming event and, then, passes on the operative content to the *essential model* for processing. Such a *routing policy* is defined as the formalization of the set of *Interactions* linked to the *Component* inside the *Routing Process*. So, the *Component* (formalized in a *routing model*) depends equally on its operative description (i.e., the internal operation formalized in the attached *essential model*) and the process structure (i.e., the related *Interactions* formalized in the *routing policy*).

Some of the advantages of employing RDEVS for the M&S of routing processes are that *i)* the modeler does not need to dip down to DEVS itself to add routing functionality to the models, *ii)* existing atomic models can be used to structure routing processes, *iii)* RDEVS models can be combined with DEVS models to define complex M&S scenarios where routing processes interact with other types of phenomena, and *iv)* DEVS simulators can be used to execute RDEVS models. Moreover, due to the conceptual foundation of routing processes as constrained network models, the UML diagrams presented in Section 2 can be combined with Model-Driven-X (MDX) and Model-Based (MB) approaches to get platform-specific implementation models.

The MDX methodologies are widely used in the Software Engineering field. For example, the Model-Driven Engineering (MDE) approach is based on several principles that involve the concepts of model, metamodel, meta-metamodel, and model transformations to provide a process that enables the automated development of a system [Cetinkaya et al. 2011]. Even when MDX approaches have been used in M&S, MB paradigms (such as MB System Engineering and MB Simulation) are also frequently used to get solutions to M&S issues [Kapos et al. 2014; Neto et al. 2018]. Such MB approaches use MDX practices pragmatically. This is, the models are important, but they do not necessarily drive the development process. In this context, in [Blas and Gonnet 2021] the authors employ MDE as a vehicle for designing and implementing a M&S software tool that provides RDEVS-based solutions for generic routing processes. They use a metamodel to abstract the routing process definition into a graphical representation based on nodes and links to describe its structure. Such a structure is used as a domain model description for building (automatically) an RDEVS implementation of the process without requiring any interaction with the modeler. Then, the modeler can define a routing process as a graph and then, get a RDEVS simulation model implementation for such a process without having programming skills.

Following the same approach, in this paper, we propose a Context-Free Grammar (CFG) to define the structure of routing processes using the constrained network model depicted in Figure 1(b). Since such grammar will be added to the M&S software tool presented in [Blas and Gonnet 2021] (developed as a plug-in for Eclipse IDE), Eclipse technology was used to implement its software modules. This will allow modelers to specify textually a routing process as a network and then, get the corresponding RDEVS simulation model implementation.

4. The Constrained Network Model Grammar

If a routing process is correctly defined as a constrained network model, then RDEVS simulation models can be automatically obtained following the formalization guidelines described in Section 3.1. In this context, we propose to use the network model that supports the routing definition (i.e., Figure 1(b)) to validate the semantic correctness of a process. The definition of such a process is obtained from the model instantiation. Such an instantiation is performed through the syntactical analysis of a natural language description called RDEVSNL. A CFG is introduced for the syntax. Hence, once the modeler has defined a natural language representation of a routing process, our strategy is to analyze the syntax of such a representation to get an instance of the model that can be validated using Figure 1(b) and Table 1.

The following subsections present both syntax and semantics that compose the grammar, following the example described in [Blas and Gonnet 2021] as proof of concept. Figure 2 shows the routing process to be defined. Icons depict machine types.

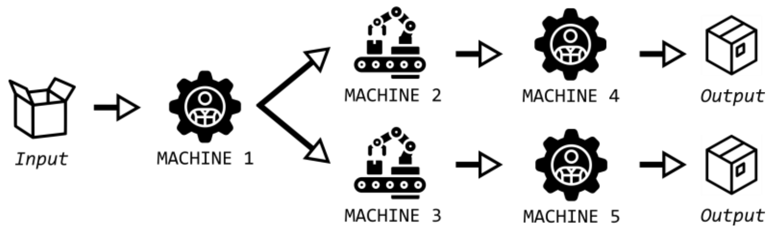
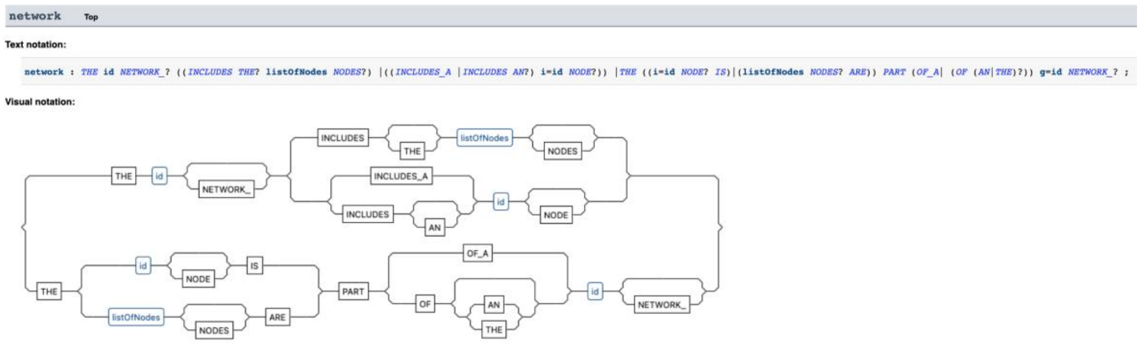


Figure 2. Industrial routing process used as proof of concept (adapted from [Blas and Gonnet 2021]).

4.1. The Syntax: RDEVSNL

The RDEVSNL grammar abstracts the routing process definition into a textual representation based on nodes and links to describe its structure. The CFG of RDEVSNL was specified and implemented using ANTLR4 (i.e., a parser generator) and ANTRL4 IDE Eclipse Plugin for ANTLR 4 [Parr 2021]. We have defined two versions of the RDEVSNL: English and Spanish. Hence, the modeler can define routing processes in both languages. We have also implemented an Editor for such a CFG as an Eclipse plugin. A RDEVSNL specification file has the extension *.rdevsnl*. The language is defined by the modeler during the creation process.

In the RDEVSNL English specification, there are three primary building blocks: *network*, *materializes*, and *edges*. Figure 3 illustrates the production rule and the syntax diagram of these three main nonterminal symbols. The *edges* symbol is partially included for space reasons. The complete definition can be found here. Furthermore, Figure 4 illustrates an example of the RDEVSNL Editor. Such an example depicts a possible specification of the routing process illustrated in Figure 2.

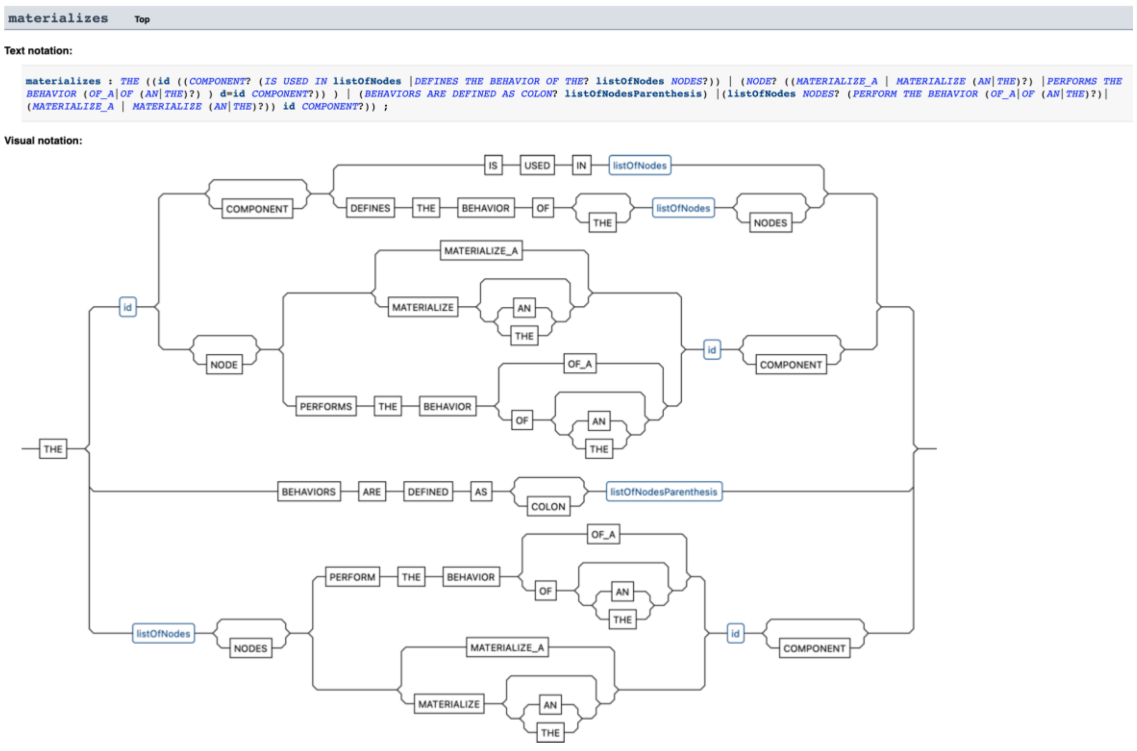

network Top

Text notation:

```
network : THE id NETWORK_? ((INCLUDES THE? listOfNodes NODES?) |((INCLUDES_A |INCLUDES AN?) i=id NODE?)) |THE ((i=id NODE? IS)|(listOfNodes NODES? ARE)) PART (OF_A| (OF (AN|THE)?)) g=id NETWORK_? ;
```

Visual notation:

(a)

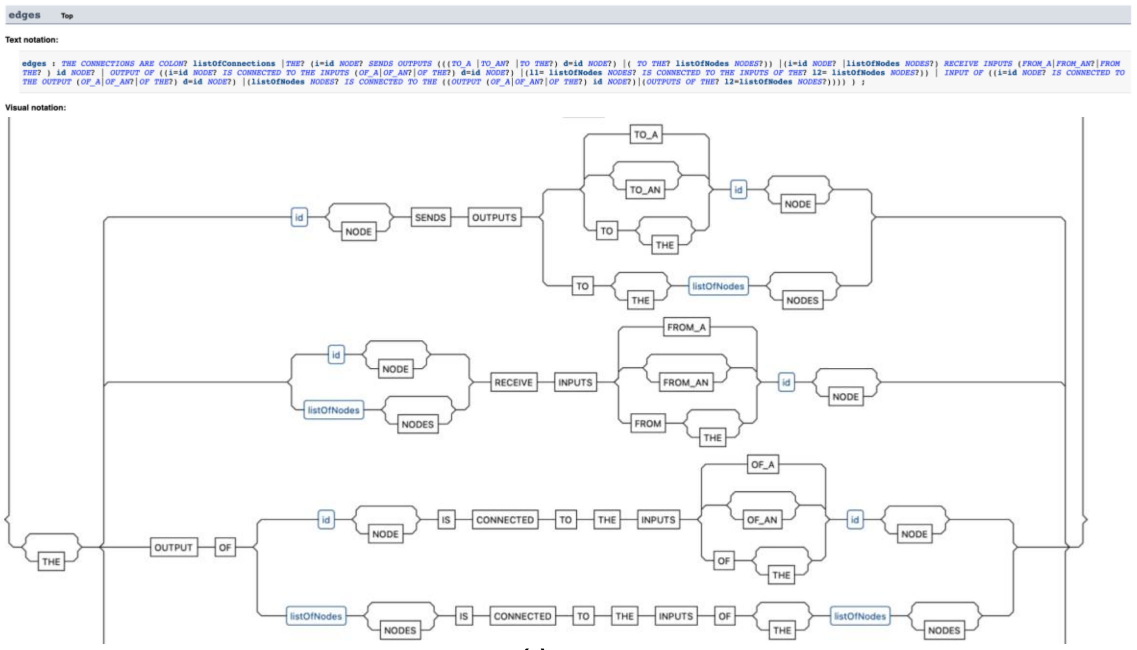

materializes Top

Text notation:

```
materializes : THE ((id ((COMPONENT? (IS USED IN listOfNodes |DEFINES THE BEHAVIOR OF THE? listOfNodes NODES?)) | (NODE? ((MATERIALIZE_A | MATERIALIZE (AN|THE)?) |PERFORMS THE
BEHAVIOR (OF_A|OF (AN|THE)?) ) d=id COMPONENT?)) ) | (BEHAVIORS ARE DEFINED AS COLON? listOfNodesParenthesis) |(listOfNodes NODES? (PERFORM THE BEHAVIOR (OF_A|OF (AN|THE)?)|
(MATERIALIZE_A | MATERIALIZE (AN|THE)?)) id COMPONENT?)) ;
```

Visual notation:

(b)


edges Top

Text notation:

```
edges : THE CONNECTIONS ARE COLON? listOfConnections |THE? (i=id NODE? SENDS OUTPUTS (((TO_A |TO_AN? |TO THE?) d=id NODE?) |( TO THE? listOfNodes NODES?)) |(i=id NODE? |listOfNodes NODES?) RECEIVE INPUTS (FROM_A|FROM_AN?|FROM
THE? ) id NODE? | OUTPUT OF ((i=id NODE? IS CONNECTED TO THE INPUTS (OF_A|OF_AN?|OF THE?) d=id NODE?) |(l1= listOfNodes NODES? IS CONNECTED TO THE INPUTS OF THE? l2= listOfNodes NODES?)) | INPUT OF ((i=id NODE? IS CONNECTED TO
THE OUTPUT (OF_A|OF_AN?|OF THE?) d=id NODE?) |(listOfNodes NODES? IS CONNECTED TO THE ((OUTPUT (OF_A|OF_AN?|OF THE?) id NODE?)|(OUTPUTS OF THE? l2=listOfNodes NODES?)))) ) ;
```

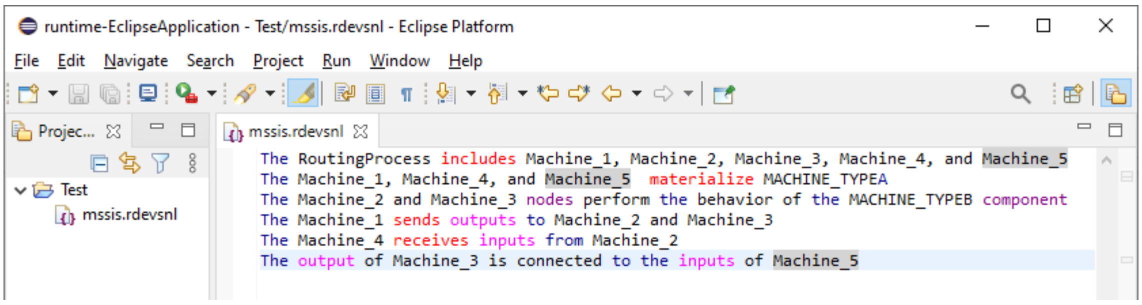
Visual notation:

(c)

Figure 3. Production rules and the syntax diagram of the three main nonterminal symbols (a) network, (b) materializes, and (c) edges.

When a RDEVSNL specification is used to structure a Routing Process, a *Network* is defined to model that process. Such a model is defined over a set of *Nodes*. Each *Node* denotes a Component (Figure 1(b)). Therefore, a *Network* is specified using an *id* and it always includes a list of nodes (Figure 3(a)). This list can be denoted in a unique specification (i.e., detailing the complete list of nodes as *listOfNodes* -as in Figure 4-) or in multiple text lines (where each *id* node is part of the *Network*). In a

Routing Process, each Component exhibits an internal operation. Such an operation is identified as a component in Figure 3(b), and it defines the behavior of a node or a list of nodes (Figure 3(b)). In Figure 1(b), *Links* define directed Interactions between *Nodes*. These interactions are specified by edges in Figure 3(c). The grammar enables the definition of these interactions in multiple ways (e.g., a node sends outputs to another node/list of nodes, a node receives inputs from a node/list of nodes, etc.) as in Figure 4.


runtime-EclipseApplication - Test/mssis.rdevsnl - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Projec...
Test
mssis.rdevsnl
mssis.rdevsnl

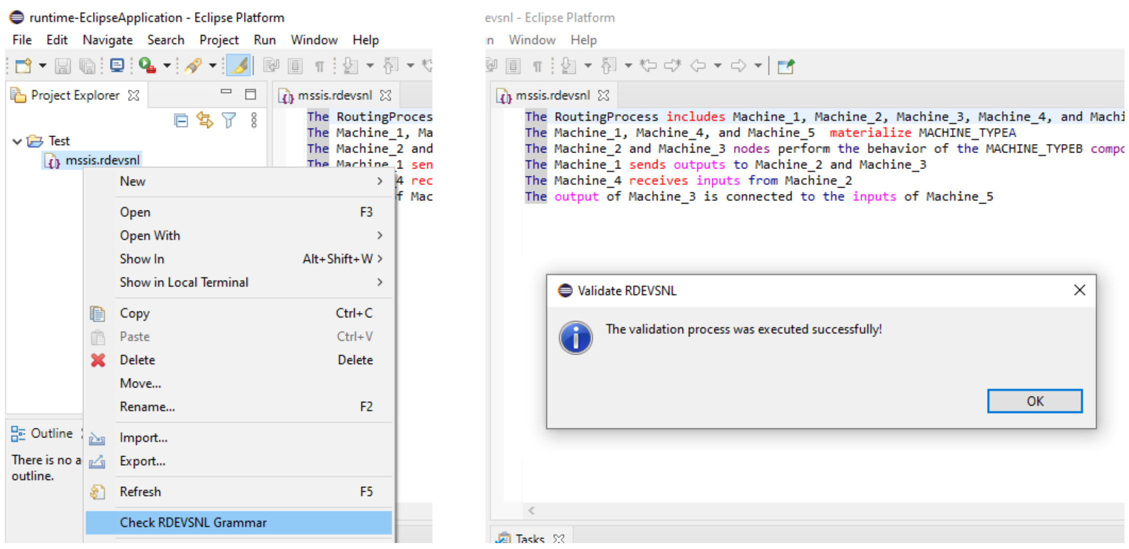
```
The RoutingProcess includes Machine_1, Machine_2, Machine_3, Machine_4, and Machine_5
The Machine_1, Machine_4, and Machine_5  materialize MACHINE_TYPEA
The Machine_2 and Machine_3 nodes perform the behavior of the MACHINE_TYPEB component
The Machine_1 sends outputs to Machine_2 and Machine_3
The Machine_4 receives inputs from Machine_2
The output of Machine_3 is connected to the inputs of Machine_5
```

Figure 4. Screenshot of the Eclipse editor implemented to support the syntax. Suggestions are provided following the language configuration. Keywords are highlighted to help the modeler during the edition.

4.2. The Semantics: The Constrained Network Model as an EMF Metamodel

With aims to validate the semantics of an RDEVSNL definition, we implement an Ecore version of the model depicted in Figure 1(b). To get such an implementation, we use the EMF project of Eclipse [The Eclipse Foundation 2021b]. This project is a modeling framework and code generation facility for building software tools and other software applications based on a structured data model. The foundational metamodel used for modeling routing processes as constrained network models (Figure 1(b)) was implemented with EMF to get a data model specification for further instantiation and validation. Since this is Java technology, such an Ecore model was embedded in the plugin. Hence, the final RDEVSNL plugin is composed of the syntax (defined using ANTLR4), the RDEVSNL editor, and the semantic model (defined as an Ecore model).

Over the CFG implemented, we add to the Editor the option to validate the description file (Figure 5(a)). When the modeler activates this validation process, the syntactical analysis (i.e., the parser) of RDEVSNL is executed over the current content of the **.rdevsnl* file. If such an analysis is successful, using the identified tokens, an instance of the Ecore metamodel is automatically created. In this instance, each element defined in the file is mapped to a concept or relationship. For example, line 1 of Figure 4 is syntactically correct. Therefore, during the validation process, the plugin creates an instantiation with the following elements: *i)* an instance of *RoutingProcess* named "RoutingProcess", *ii)* five instances of *Component* (each one named "Machine_1" to "Machine_5"), and *iii)* five relationships *isDefinedOver* to link each *Component* to the *RoutingProcess*. Then, over such an instance, the plugin runs the Ecore validation process to ensure its correctness. This validation checks the concepts, relationships, multiplicities, and OCL constraints of the metamodel over the instance (i.e., the semantic analysis). If both analyses are successful (i.e., syntax and semantic), the modeler gets the message of Figure 5(b). On the other hand, if issues are detected, a list of warnings is provided to the modeler. Then, the modeler can fix its routing process description and check again.



```
The RoutingProcess includes Machine_1, Machine_2, Machine_3, Machine_4, and Machi
The Machine_1, Machine_4, and Machine_5  materialize MACHINE_TYPEA
The Machine_2 and Machine_3 nodes perform the behavior of the MACHINE_TYPEB compo
The Machine_1 sends outputs to Machine_2 and Machine_3
The Machine_4 receives inputs from Machine_2
The output of Machine_3 is connected to the inputs of Machine_5
```

The validation process was executed successfully!

Figure 5. Validation process of the routing process description. (a) At the left, the Java editor option used to run the CFG validation. (b) At the right, the confirmation message about the correctness of the routing process definition.

5. Conclusions and Future Works

In this paper, we have introduced a grammar based on a constrained network model to address the definition of discrete-event simulation models for routing processes through the RDEVS formalism. The core of RDEVS is the formalization of the set of elements that composes a routing process using a conceptual modeling approach. Then, the RDEVS formalism levels out the complexity of routing process specification to reduce the modeling effort. Using a constrained network model allows identifying all the elements required to define a routing process and, therefore, the RDEVS formalism can be further used to formalize these elements. Here, we have presented the first step in this direction.

We have shown that routing structures can be defined in natural language using a new grammar based on a constrained network model as support. As in the graphical definition proposed in the previous work, since all the data regarding the routing process structure can be obtained from the network abstraction, RDEVS network and routing models can be fully specified from this description. Then, the modeler only needs to complete the simulation model with the internal operation of components.

The network model basis used allows improving the routing process representation to get a deeper analysis of the associated simulation model. Much of the theoretical wealth of network analysis consists of characterizing network structures (e.g., small-worldness) and node positions (e.g., centrality). Hence, network analysis can be performed over the RDEVS simulation models to get information regarding the structure of the simulation process.

Future work is devoted to the translation of the routing process described in the network model instance (obtained from the grammar) to the RDEVS simulation models (following the definitions provided in Section 3.1). Such a translation process will be developed as the one already implemented in [Blas and Gonnet 2021]. Furthermore, we will use the existing graphical representation to display the natural language description as a graph. Hence, the M&S software tool will support both graphic and text definitions.

References

Blas, M. and Gonnet, S. (2021). Computer-aided Design for Building Multipurpose Routing Processes in Discrete Event Simulation Models. In *Engineering Science and Technology, an International Journal*, vol. 24, pages 22–34.

Blas, M., Gonnet, S. and Leone, H. (2017). 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*, pages 774-785.

Borgatti, S. P. and Halgin, D. S. (2011). On network theory. In *Organization Science*, vol. 22(5), pages 1168-1181.

Cetinkaya, D., Verbraeck, A. and Seck, M. (2011). MDD4MS: A Model Driven Development Framework for Modeling and Simulation. In *Proceedings of the 2011 Summer Computer Simulation Conference*, pages 113–121.

Kapos, G., Dalakas, V., Tsadimas, A., Nikolaidou, M., and Anagnostopoulos, D. (2014) Model Based System Engineering using SysML: Deriving Executable Simulation Models with QVT. In *Proceedings of the 2014 IEEE International Systems Conference*, pages 531–538.

Neto, V., Manzano, W., Kassab, M., and Nakagawa, E. (2018). Model-based Engineering & Simulation of Software-Intensive Systems-of-Systems: Experience Report and Lessons Learned. In *Proceedings of the 2018 European Conference on Software Architecture*, pages 1–7.

Newman, M., Barabasi, A.-L., and Watts, D. J. (2006). The Structure and Dynamics of Networks. Princeton University Press.

OMG (2002). Meta Object Facility (MOF) Specification, Version 1.4.

Pan, W. (2011). Applying Complex Network Theory to Software Structure Analysis. In *International Journal of Computer and Systems Engineering*, vol. 5(12), pages 1634-1640.

Parr, T. (2021). ANTLR. Available at https://www.antlr.org/ (accessed 28th June 2021).

The Eclipse Foundation (2021a). Eclipse. Available at https://www.eclipse.org/ (accessed 24th June 2021).

The Eclipse Foundation (2021b). Eclipse Modeling Project. Available at https://www.eclipse.org/modeling/emf/ (accessed 29th June 2021).

Wen, L., Kirk, D. and Dromey, R. G. (2007). Software Systems as Complex Networks. In *Proceedings of the 2007 IEEE International Conference on Cognitive Informatics*, pages 106-115.

Zakari, A., Lee, S. P., and Chong, C. Y. (2018). Simultaneous Localization of Software Faults based on Complex Network Theory. In *IEEE Access*, vol. 6, pages 23990-24002.

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