Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

FROM ABSTRACTION TO IMPLEMENTATION: IMPROVING THE RDEVS MODELING AND SIMULATION THROUGH A DOMAIN MODELING SPECIFICATION

María Julia Blas
Silvio Gonnet

Instituto de Desarrollo y Diseño INGAR
CONICET-UTN
Avellaneda 3657
Santa Fe, 3000, ARGENTINA

ABSTRACT

The Routed DEVS (RDEVS) formalism provides a feasible solution to modelers for easily build discrete-event simulation models for routing processes. By employing three types of models, the formalism provides an appropriate separation of concerns in terms of the routing structure, the possible paths and the component behaviors. This paper presents the usefulness of graphical Modeling and Simulation (M&S) for supporting the implementation of routing process simulation models thought their abstraction to a graph model. These simulation models are formalized using RDEVS formalism. The methodological proposal is focused in the development of a M&S software tool based on a combination of data metamodels and formalizations rules. The final software tool was developed as a plugin for Eclipse IDE with aims to take advantage of existent M&S software. One of the main benefits obtained when modeling an abstraction with the proposed M&S tool is a significant reduction of formalization and implementation times.

1 INTRODUCTION

The conceptual models generally describe the structure and the behavior of the system independent from the implementation details (Cetinkaya et al. 2010). The success of the modeling task depends on the modeler ability to get an appropriate level of abstraction in the design. Although effective conceptual modeling is a vital aspect of a simulation study, it is probably the most difficult and least understood (Law 1991). The design of the simulation model impacts all aspects of the study, in particular the data requirements, the speed with which the model can be developed, the validity of the model, the speed of experimentation and the confidence that is placed in the model results (Robinson 2008). Good modeling enables an appropriate separation of concerns that improves quality properties of the final simulation models, such as modifiability and maintainability. In this sense, the Modeling and Simulation (M&S) software products should provide not only simulation capabilities but, also, they should support new modeling strategies that improve the modeling tasks (e.g. number of simulation models to be developed, time to spend for getting correct implementations, modeling complexity, M&S knowledge required prior perform the modeling, etc.).

Well-designed M&S software products should support reuse of existing (software) components (Dalle et al. 2009). Moreover, two desirable properties of any M&S software product are *low coupling* and *high cohesion* among software components. The *low coupling* is desirable because: *i)* fewer interconnections among software modules reduce the chance that changes in one module cause problems in other modules (i.e. enhances reusability), and *ii)* fewer interconnections among modules reduce the developer time in understanding the details of other modules (Page-Jones 1980). In this context, any M&S software tool has to, al least, provide two distinct set of modules: software modules for supporting the modeling, and software modules for supporting the simulation execution. On the other hand, *cohesion* is an important attribute corresponding to the quality of the abstraction captured by the software module under consideration. Good

abstractions typically exhibit *high cohesion* (Hitz and Montazeri 1995). Thereby, each software module defined as part of the M&S software product, needs to fully capture an explicit abstraction of the M&S task.

Nowadays, there are multiple software tools and simulators for DEVS models (Van Tendeloo and Vangheluwe 2017). Most M&S DEVS tools support graphical modeling capabilities. For example, PowerDEVS (Bergero and Kofman 2011) integrates different software modules with aims to provide a graphical modeling environment, an atomic model editor, and a code generator in a M&S DEVS tool. In the same direction, the graphical modeling environment called CD++Builder (Bonaventura et al. 2013) can be used to create models for CD++ (Wainer 2002). DEVSimPy (Capocchi et al. 2011) offers a graphical modeling environment for coupled models. A similar approach is applied in Virtual Laboratory Environment (VLE) (Quesnel et al. 2007) where atomic models are written in C++ and coupled models can be created using either the graphical environment or by manually writing XML files. However, all these approaches are centered in building graphically models that have been already designed by some modeler (at least, at conceptual level).

From the traditional point of view, a good conceptual model lays a strong foundation for successful simulation modeling and analysis (Robinson et al. 2010). Over such interpretation, conceptual modeling serves as a bridge between problem owner and simulation modeler. In this context, the *progression of abstraction to implementation* presented by Zeigler et al. (2018) can be used to study how abstractions can lead to well-defined implementations. An *abstraction* focuses on an aspect of reality and, almost by definition, greatly reduces the complexity of the reality being considered. Then, *formalization* makes it easier to work out implications of the *abstraction* and implement in reality. Finally, *implementation* can be considered as providing a concrete realization of the *abstraction*. The use of the progression *abstraction-formalization-implementation* allows to develop new types of M&S software products centered in new model representations.

In this paper, we propose a M&S software tool implemented as a plugin for Eclipse Integrated Development Environment (IDE) (The Eclipse Foundation 2020a) supporting the description of routing process simulation models thought their abstraction to a graph model. We employ Routed DEVS (RDEVS) as formalization mechanism for such simulation models. The RDEVS formalism was designed as a DEVS extension that improves the discrete event models that solve problems centered in routing processes (Blas et al. 2017). A *routing process* is defined as "the part of a modeling scenario where the components need to interact among them by distinguishing the event sources and destinations in order to ensure their transference into the right model". When building a simulation model for solving a *routing process*, the modeler is solving a *routing problem*. Hence, the core of RDEVS formalism is the formalization of the set of elements that composes a *routing process* definition following a conceptual modeling point of view. Then, the RDEVS formalism levels out the complexity of *routing problems* specification to reduce the modeling effort. This allows the use of the RDEVS formalism as a "layer" above the DEVS formalism that provides routing functionality without requiring the user to "dip down" to DEVS itself for any functions (Zeigler 2018). In this context, our aim is: *i)* to offer a graphical environment for modeling RDEVS formalism using a standardized graph description, and *ii)* to be able to generate Java code for such RDEVS models in a way that they can be executed in DEVS simulators.

The remainder of this paper is structured as follows. Section II describes the principles of RDEVS formalism detailing the simulation models and their structural dependencies. It also includes the definition of the *abstraction-formalization metamodel* used as conceptual view of the proposal. Section III presents the M&S software tool designed using the *abstraction-formalization-implementation* approach. Section IV briefly describes the main benefits of having the M&S software tool for building routing process simulation models based in RDEVS formalism. Finally, Section V is devoted to conclusions and future work.

2 RDEVS FORMALISM

The RDEVS formalism defines three types of models: *essential*, *routing* and *network* (Blas et al. 2017). These models allow the formalism to provide an appropriate separation of concerns over the routing process description (i.e. the structure and routing paths) and the routing component behaviors.

The *essential model* specifies a discrete-event simulation model that exhibits the behavior of an elemental routing component that will be used as part of a routing process description. The guidelines for building an *essential model* should be defined by a domain expert (because such model refers to a domain specific component). Formally, a RDEVS essential model is defined as a DEVS atomic model (Zeigler et al. 2018).

The *routing model* defines a discrete-event simulation model that acts inside the routing process. Its definition employs a routing process component as operational description of its own behavior. Hence, the *routing model* definition embeds an *essential model*. The same *essential model* can be embedded in several *routing models*. Also, the *routing model* definition requires setting the *routing policy*. The *routing policy* attached to each *routing model* is defined at design time and cannot be changed during the simulation execution. However, the same *routing policy* can be used in distinct *routing models* (that, at the same time, can be based in distinct *essential model*). The *routing policy* definition includes an identifier used to distinguish the set of *routing models* that build the routing process. *Routing models* employ these identifiers to decide how to treat the input events (accept or reject) and how to route the output events. Therefore, the *events* that flow over the *routing models* that compose a routing process are defined as *events with identification*. An *event with identification* is recognized by its sender information (i.e. the identifier of the *routing model* that created the event through its output function) and its feasible receptors. The feasible receptors of an *event with identification* are obtained from the *routing function* that compose the *routing policy*. Therefore, the execution of the routing process is build-in inside the RDEVS models.

Finally, the *network model* describes a complex discrete-event simulation model that has a primary goal that involves the resolution of a routing problem. Its definition includes a set of *routing models* and the couplings among them. Such couplings are detailed as all-to-all connections in order to left the routing task to the *routing policies* detailed in the *routing models*. The *network model* specification also involves two special translation functions used to link distinct networks: input translation function and output translation function. These functions allows matching *events with identification* produced by different routing process. Therefore, *network models* are designed to interact with other *network models* or, simply, with DEVS models (atomic or coupled). The output events produced by a *network model* are events sent "everywhere". The simulation model that, eventually, consumes such events must decide how to route these messages. Hence, the combination of distinct types of simulation models depends on the problem final goal allowing to build powerful simulation models that exploit the benefits of each formalism according to the problem characteristics.

By embedding *essential models* into the *routing models*, the RDEVS formalism improves the design phase in two distinct ways: *i)* behavioral descriptions are only required for the domain-specific routing components (the routing behavior is build-in as part of the RDEVS models), and *ii) routing policies* are isolated from behavioral descriptions (allowing to reuse the *essential models* in new *networks models*). Therefore, the RDEVS models allows the modeler to explicitly define the *routing policy* without specifying any behavioral description attached to it. Such routing information is detailed inside the *routing model* as part of its own definition in order to authenticate senders and receivers prior executing the (linked) *essential model*.

The RDEVS formalism is designed for level out the modeling effort of routing problems providing an easier modeling solution that employs a set of simulation models defined in terms of the main elements involved in routing processes. RDEVS models can be used as predefined simulation modules (i.e. patterns) that require specifying: a behavior (*essential model*), a routing component with its own routing policy (*routing model*) and the interactions among such components (*network model*). In this context, the aim of RDEVS models is isolate a routing process inside the *network model* and capture the explicit information required to redirect the events flow over the *routing models* in the *routing policies*. That is, there is no universal routing policy to manage the global structure of the simulation model. Each *routing model* takes its own decisions (thought the *routing policy*) on how routing the input and output events. The modeler does not need to explicitly define any additional behavior to manage the routing process during design phase.

Then, the RDEVS formalism provides a feasible solution to modelers for easily build discrete-event simulation models for routing processes. Since RDEVS is a subclass of DEVS, the RDEVS models can be executed using DEVS simulators. Then, RDEVS models can be implemented employing existent implementations of DEVS. In (Blas et al. 2017) a first description of the *RDEVS software framework* was presented.

In addition to the reusable functions provided in libraries, *frameworks* provide "flows of control" (Johnson and Foote 1988). *Frameworks* shall ease / speed up the development of software from the domain they are created for (Madsen 2003). Therefore, a *framework* may be built on top of a set of libraries and might be used to create more specialized solutions. This is the case of the *RDEVS software framework* that allows modeling and executing RDEVS models in Java using the features provided by DEVSJAVA (ACIMS 2005) and DEVS Suite (ACIMS 2009). DEVSJAVA is a M&S tool implemented in Java that supports characterizing models in DEVS formalism. On the other hand, the DEVS Suite Simulator is itself an extension of DEVSJAVA that supports visual design of experiments and introduces simulation data visualization (Kim et al. 2009). Through the extension of DEVSJAVA and DEVS Suite Simulator, the *RDEVS software framework* provides a solid solution for building executable models that support RDEVS formalism.

Hence, the use of the *RDEVS software framework* enhances the development of RDEVS simulation models in Java programming language.

2.1 Routing Process Simulation Models from Graph Abstractions

An abstraction focuses on an aspect of reality and, almost by definition, greatly reduces the complexity of the reality being considered (Zeigler et al. 2018). In this context, graph models (defined over a set of predefined components) can be used as abstraction of routing processes (reality).

When a routing process is modeled as a graph, the nodes of the graph represent domain-component instances and the edges of the graph represent the connections (i.e. the relationships) among those instances (i.e. the nodes). Under this representation, the domain components are conceptualized as nodes of the graph that share a behavioral operation.

Figure 1 shows a UML class diagram that describes the main elements that compose a graph model. A *Graph* is *Composed by Nodes* and *Edges*. In this case, a *Graph* is *Composed by*, at least, two *Nodes* and it *Includes*, at least, one *Edge*. Both associations (i.e. *Composed by* and *Includes*) are modeled as UML compositions. The *Nodes* are linked by *Edges*. An *Edge* is defined as an ordered pair of *Nodes* according to the mandatory associations named *Starts at* and *Ends at* (both cases are modeled with multiplicity 1). Finally, a *Node Instantiates* a *Component* that have a *behavior* described over a *domain procedure*. The *Instantiates* association between *Node* and *Component* is mandatory for *Node* (multiplicity 1). However, the same *Component* can be used as support of several *Nodes* (multiplicity 1..*).

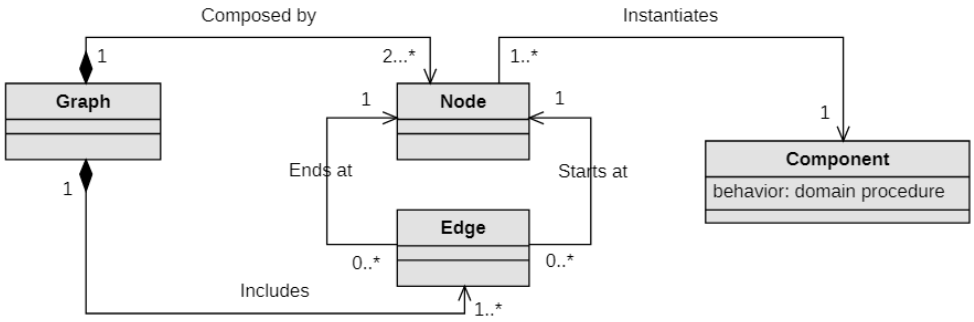

Figure 1: Simplified UML class diagram that depicts the main concepts and relationships required for instantiate a graph model that represents a routing process (*abstraction metamodel*). The routing process needs to be based in a predefined set of components that hold a well-known behavior defined in a domain procedure.

The *abstraction metamodel* depicted in Figure 1 simplifies the graph domain using a set of concepts and relationships that ensure building an appropriate graph model. The main advantage of employing a graph model as abstraction of the routing process domain is that the existent metrics designed for such models can be used as support measures for evaluate the complexity of the simulation models obtained from it.

Following the abstraction, a formalization makes it easier to work out implications of the abstraction and implement in reality (Zeigler et al. 2018). When a simulation model is designed for a routing process, the RDEVS formalism can be used as formalization language. Since the formalism provides a solution to modelers for easily build such models, the RDEVS models allows to *formalize* the routing process modeled as a graph with aims to get discrete-event simulation models from such abstraction.

Then, each RDEVS model can be seen as the formalization of some component defined in Figure 1 as follows:

- the essential model *formalizes* a component (because the essential model is designed to exhibit the behavior of an elemental routing component),
- the routing model *formalizes* the nodes of the graph (because the routing model is designed to act as part of the routing process), and
- the network model *formalizes* the graph (because the network model is designed to represent the overall routing process).

Figure 2 extends the UML class diagram depicted in Figure 1 with the *formalization metamodel* (i.e. concepts highlighted in yellow) employing the *formalizes* stereotyped relationship (i.e. association highlighted in blue). Through this special relationship, the *formalization metamodel* refines the concepts of the *abstraction metamodel* with aims to explicitly define the simulation model structures. For example, the *Node* formalization through the *Routing Model* includes the node decomposition in terms of its *Routing Policy* and *Input* / *Output Ports*. A similar approach is used in the *Network Model* that formalizes the *Graph*. In this case, the *Input* and *Output Translation Function* are included to formalize the *Node* (or *Nodes*) where the event flow should start and end, respectively.

A set of OCL constraints is added to the UML class diagram in order to maintain the traceability between *abstraction* and *formalization metamodels*. This traceability is related to the correctness of the formalism. Even when RDEVS models can be used to support the design of routing processes, the graph model used as abstraction must ensure a set of structural properties prior is formalization with RDEVS. Such structural properties guarantee the validity of the RDEVS models.

The OCL constraints detailed in Figure 2 are attached to the *formalizes* relationship modeled between *Node* (from the *abstraction metamodel*) and *Routing Model* (from the *formalization metamodel*). These constraints should be evaluated prior getting the *Routing Model* that formalizes a *Node*. For example, a *Routing Model* cannot be obtained for an isolated *Node* (because, by definition, all models that compose a *Network Model* needs to be connected to each other). Then, before obtaining such simulation model, the OCL invariant named *isNotIsolated* should be evaluated over the *Node*.

From a conceptual point of view, a *Coupling* can be seen as a formalization of an *Edge*. However, the diagram depicted in Figure 2 does not include such formalization as an effective relationship. This is because the *Couplings* in RDEVS models are structured by the *Network Model* definition as all-to-all connections (in order to left the routing task to the *Routing Policies*). When a *Routing Model Embeds* an *Essential Model*, the *Routing Policy* must be defined. Such *Routing Policy* is used to route the input/output events that flows to/from the actual *Routing Model* from/to the other *Routing Models* that compose the *Network Model*. Therefore, the *Routing Policy* is *Defined over* a set of *Routing Models* (at least, one *Routing Model* given by the multiplicity 1…*) using as foundation the *Edges* that link the *Nodes* that abstract them. Moreover, all *Routing Models* included in a *Network Model* must be attached to, at least, one *Routing Policy* (multiplicity 1…*) - because they cannot be obtained from isolated *Nodes*. Hence, the *Edges* are used as guidelines for building *Routing Policies* but not as abstraction of the *Couplings*.

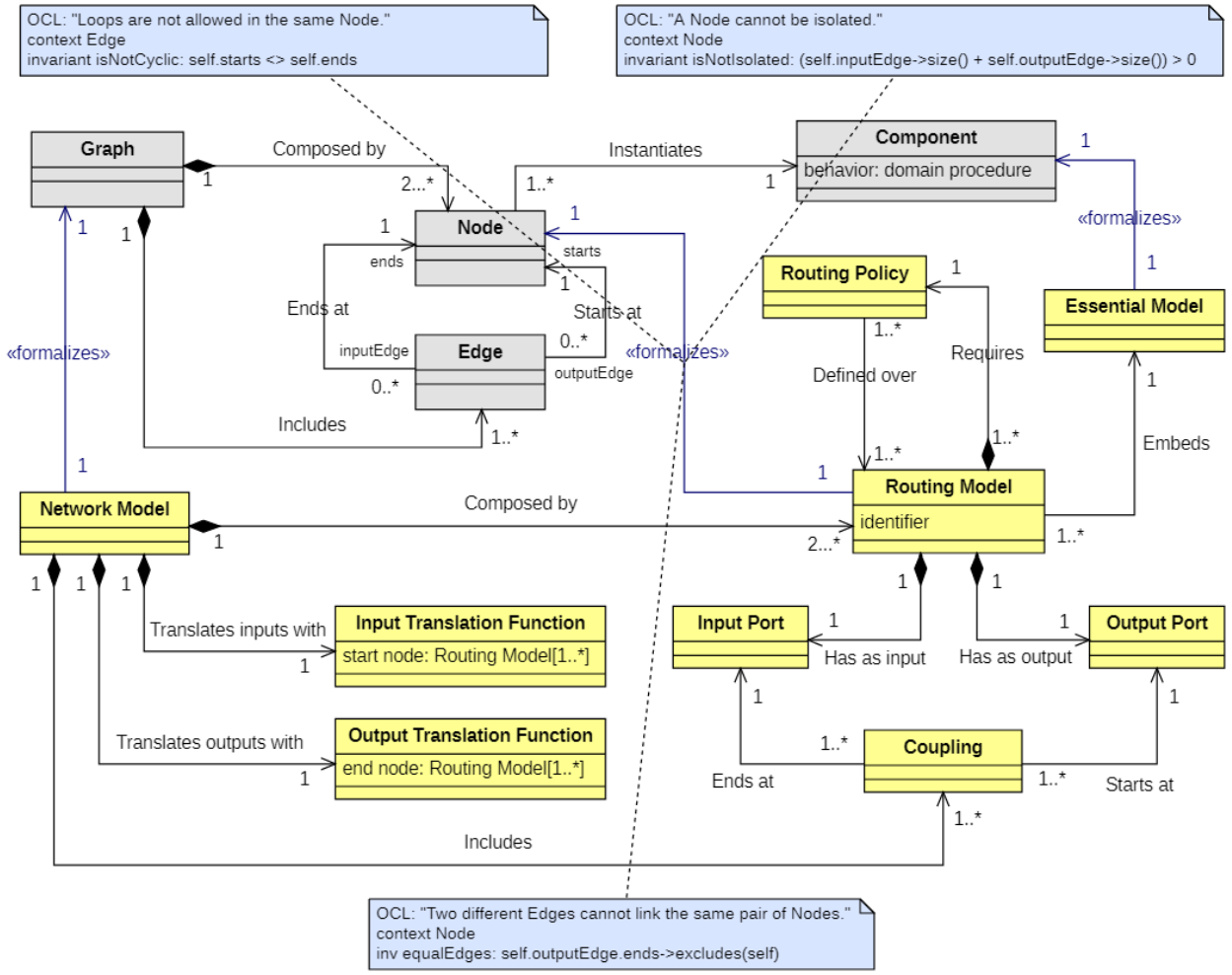

Figure 2: UML class diagram that links the *abstraction metamodel* with the RDEVS specification through the *formalizes* relation. The associations that holds the same name and multiplicity in both metamodels describe the same dependency between elements. The *Embeds* association (from the *formalization metamodel*) is the *materialization* of the *Instantiates* association (from the *abstraction metamodel*). The OCL constrains are used to ensure the correct design of RDEVS models.

Under the *abstraction-formalization* representation, the routing process problem is modeled with a strict separation of concerns between the model primary goal (i.e. the domain procedures) and the routing process itself. However, in order to get a concrete realization of it, such representation must be implemented. Then, the implementation can be considered as providing a concrete realization of the abstraction thought the formalization. An available implementation of RDEVS models is given in the software framework initially presented in (Blas et al. 2017).

The main classes designed and implemented as part of the *RDEVS software framework* are the following:

- *EssentialModel.java*, *RoutingModel.java*, and *NetworkModel.java* to define the simulation models that build RDEVS formalism,
- *RoutingFunction.java* and *RoutingFunctionElement.java* to define the *Routing Policy* required as part of the *Routing Model* definition, and

- *TranslationFunction.java*, *InputTranslationFunction.java* and *OutputTranslationFunction.java* to define, respectively, the transformations from *events* / *events with identification* to *events with identification* / *events*.

The Java classes listed above can be seen as potential *extension points* of the software framework for building an explicit RDEVS implementation. In *software engineering*, an *extension point* is the definition of the provided interface for extensions (Klatt and Krogmann 2008). That is, an extension itself is an implementation according to an *extension point* (equal to an implementation of a software component). Therefore, the software framework includes *extension points* configured for designing explicit instances of RDEVS models as reusable software components slated for executing the routing process simulation (already defined in the framework) without any other consideration. In this context, each one of the concepts defined as explicit *formalization* of the routing process *abstraction* can be *implemented* as an extension developed over an *extension point* of the software framework (i.e. a new Java class based on an existent one).

Following this approach, the next section introduces the plugin for Eclipse IDE developed to integrate the *abstraction-formalization metamodel* with the *RDEVS software framework* (already developed in Java). The goal of this plugin is provide a single object-oriented software M&S tool for building routing process simulation models from graph abstractions.

3 M&S GRAPHICAL SOFTWARE TOOL FOR BUILDING ROUTING PROCESSES FROM ABSTRACTION TO IMPLEMENTATION

Graphical modeling is an aspect which is used successfully (Ören 2007). Among the advantages of a graphical software tool are: *i)* easy-to-use: in graphical modeling software, the modeling is performed by manipulating graphical elements and their connections; *ii)* fast modeling solutions: graphical modeling software allows the development and solution of complex simulation models rapidly with limited M&S background; *iii)* well-defined simulation models: if code generation is implemented over the graphical modeling, the final simulation models will always be well-defined in terms of the related simulation formalism; and *iv)* standardized simulation models: thought the employment of a set of well-defined graphical components, the graphical modeling software ensures standardized designs (Nikolaidou et al. 2008; Touraille et al. 2011; Bonaventura et al. 2013; Wainer 2017).

With aims to build a M&S software tool that provides an full solution for building simulation models for routing processes, a graphical M&S plugin for Eclipse IDE was developed. The core of this plugin is the *abstraction metamodel* depicted in Figure 1.

Figure 3 shows the plugin architecture over a layered design pattern build over five software modules (where the modular code runtime platform is Eclipse IDE). The layered architecture employed as structure for the M&S software allows to get reusable software modules and, at the same time, modularity in each M&S level. By employing specific plugins in the development of each module, the performance of supporting tools is taken into advantage.

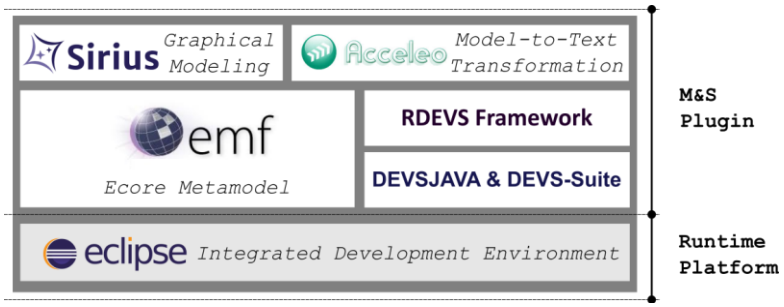

Figure 3: Layered software architecture of the M&S plugin.

Besides employing DEVSJAVA, DEVS Suite-Simulator and the RDEVS software framework, the M&S plugin uses other software modules for building the routing process representation (i.e. the abstraction, formalization and implementation). With aims to ensure a full compatibility with the underlying platform, the development of the M&S software employed several plugins of Eclipse IDE. The Eclipse Modeling Framework (EMF) (The Eclipse Foundation: Eclipse Modeling Project 2020) was used for building the foundational metamodel (*abstraction metamodel*) required as support mechanism of the plugin. The Sirius project (The Eclipse Foundation 2020b) was employed for developing a graphical representation of the foundational metamodel. Such graphical representation allows building routing process representations (abstraction metamodel instances) employing a set of graphical elements that depict the concepts and relationships detailed in the abstraction metamodel. Finally, the Acceleo development tool (The Eclipse Foundation 2019) was used for translating the abstraction model (i.e. the one obtained as an instance of the *abstraction metamodel*) to the Java implementation of RDEVS (using the formalizations defined in the *formalization metamodel*).

3.1 Ecore Model

The EMF project is a modeling framework and code generation facility for building software tools and other software applications based on a structured data model (The Eclipse Foundation: Eclipse Modeling Project 2020).

The foundational metamodel used for modeling the routing process was implemented with EMF in order to get a data model specification described in XMI. This implementation provides the Ecore model required for building the other software modules that compose the plugin. The core of such metamodel was the *abstraction metamodel*. However, a few changes were introduced to the original metamodel with aims to exploit the EMF capabilities. First, the association named *Instantiates* was renamed to *Materializes* in order to remove the notion of *instance* from the modeling tasks. Also, new conceptual modeling elements were added. These elements include:

- the attribute *name* in concepts *Graph*, *Node* and *Component* to provide a further identification of their instances, and
- the concept *Abstraction Description* as a container of the set of elements that compose an explicit abstraction.

The *Abstraction Description* concept was introduced in the metamodel detailing the following associations: *i)* an *Abstraction Description Describes* a *Graph* (i.e. a mandatory composition), and *ii)* an *Abstraction Description Uses* a set of *Components* (at least, one *Component* per instance).

Moreover, since Ecore provides the capability to include OCL constrains over the models, the OCL constrains required to get a formalization from the abstraction were also included over the foundational metamodel. In this way, the abstraction model (instance of the foundational metamodel) can be validated prior its formalization. Then, an explicit and valid abstraction of a routing process based in the graph model can be instantiated using the Ecore model defined with EMF.

3.2 Sirius Graphical Definition

Sirius is an Eclipse project which allows to easily create a graphical modeling workbench by leveraging the Eclipse Modeling technologies (The Eclipse Foundation 2020b).

Using as basis the Ecore model implemented with EMF, Sirius was employed to define graphical representations for each element included in the model. Figure 4 shows a screenshot of the final graphical software tool developed with Sirius. In the example, there are four *Components* (Machine Type A, B, C and D) and five *Nodes* (Machine #1 to #5). Each *Node materializes* an specific *Component* (e.g. the *Node* named “Machine #1” *materializes* the *Component* named “Machine Type A”). Finally, the *Nodes* are linked by *Edges* (grey arrows). These *Edges* depict the event flows allowed among the (future) simulation models (to be implemented with RDEVS).

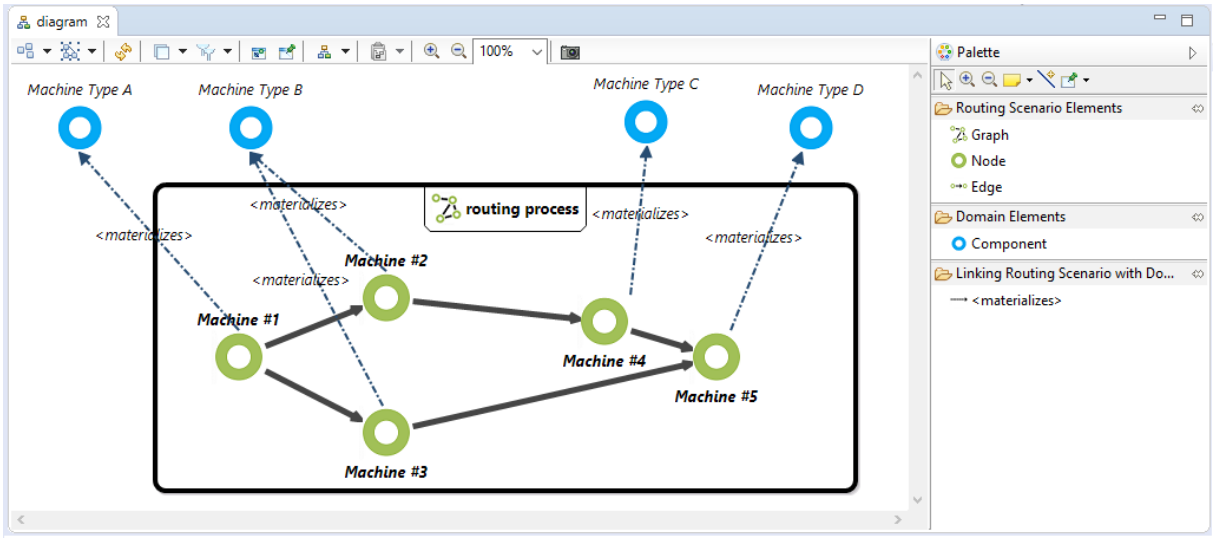

Figure 4: Screenshot of the graphical M&S software tool for modeling the routing process abstraction. The modeler drag and drop the elements available in the palette (right side of the screen) to build the routing process abstraction.

Besides providing a graphical instantiation of the Ecore metamodel, the tool allows to validate (graphically) the abstraction. A validation button was added with aims to verify the correctness of the final model (i.e. the OCL constrains along with multiplicities and attributes uniqueness). If problems are detected, the tool shows an error message with a warning icon over the element. If the model is correct, the modeler can go directly to the *formalization-implementation* phase over the *abstraction model* instantiated.

3.3 Acceleo Model-to-Text Transformation

Acceleo is a template-based technology including authoring tools to create custom code generators. It allows to automatically produce any kind of source code from any data source available in EMF format (The Eclipse Foundation 2019).

By defining a generation model for the model-to-text transformation, the elements graphically defined in the *abstraction model* are navigated in order to apply the required formalizations. These formalizations allows to get the Java source code for implementing the RDEVS simulation models. Therefore, their design goal is defining the type of RDEVS model required for the abstraction elements as follows:

- each *Component* included in the *Abstraction Description* is formalized in an *essential model* structure that will require (after getting the source code structure) codifying its behavioral specification (e.g. the transition and output functions);
- each *Node* included in the *Graph* is formalized in a *routing model* that encompasses the model identifier (that is defined as an unique value over the *Graph*), a pre-set of the *routing function* (given by the *Edges* detailed for the *Node* over the *Graph*) and an instance of the *essential model* obtained for the related *Component* (the one *materialized* by the *Node*);
- the *Graph* included in the *Abstraction Description* is formalized in a *network model* structure that includes all the *routing models* obtained for its *Nodes*, and that will require (after getting the source code structure) codifying the transformations needed for its *input/output translation functions*.

Following these formalizations, the M&S software tool creates the Java classes that extend the classes already implemented in the *RDEVS software framework*. The main template used in the automatic generation process is the following:

```
[template public generateElement(abstraction : AbstractionDescription)]
   [for (c : Component | abstraction.uses)]
      [generateComponentStateClass(c)/]
      [generateComponentEssentialModel(c)/]
   [/for]
   [for (n : Node | abstraction.describes.composedBy)]
      [generateRoutingFunctionClass(n)/]
      [generateRoutFunctionElementClass(n)/]
      [generateNodeRoutingModel(n)/]
   [/for]
   [generateGraphNetworkModel(abstraction.describes)/]
   [generateTranslationFunctionClasses(abstraction.describes)/]
[/template]
```

This template shows how the elements included in the *abstraction model* are navigated (through the association modeled in the *Abstraction Description* concept) to get their formalization. The `generate()` methods are used to support the Java code generation. Table 1 summarizes the set of classes obtained when a model-to-text transformation is performed over the *abstraction-formalization model*.

Finally, ones the abstraction model is translated to its implementation code, the modeler must codify the pending behaviors and translations prior executing the simulation. After such codification, the RDEVS simulation model will be executable. At this point, it is important to remark that this remaining codification is intended to defining the explicit behaviors of the domain. The overall behavior required for supporting the routing process execution is provided by the definition of RDEVS models. Hence, through the use of the M&S software tool, modeling and implementation times are reduced to the automatic translation of an abstraction model defined graphically from the domain description.

Table 1: Java classes created from the Ecore model as extension of the RDEVS software framework.

Ecore Model		**Java Implementation**	
Concept	**Attribute**	**New Class**	**"extends"** *(from RDEVS software framework)*
Component	*name*	<*name*>EssentialModel.java	EssentialModel.java
		<*name*>State.java	State.java
Node	*name*	<*name*>RoutingModel.java	RoutingModel.java
		<*name*>FunctionElement.java	RoutingFunction.java
		<*name*>RoutingFunctionElement.java	RoutingFunctionElement.java
Graph	*name*	<*name*>NetworkModel.java	NetworkModel.java
		<*name*>InputTranslationFunction.java	InputTranslationFunction.java
		<*name*>OutputTranslationFunction.java	OutputTranslationFunction.java

4 BENEFITS OF THE M&S SOFTWARE TOOL FOR RDEVS SIMULATION MODELS

When using RDEVS formalism for building discrete-event simulation models for routing process, the modeler can focus its attention to domain properties without worry about the routing implementation. The simulation models effectively built by the modeler (behaviors) are the ones that represent well-known domain elements. That is, take as example a *routing process abstraction* designed with *N* nodes. If all nodes instantiate the same component, the modeler only has to design *1* simulation model (the one that represent the behavior of such component) but, the final number of simulation models required for managing the overall routing process will be *N + 2* (one for the component and one for the graph). As opposite, if each node instantiates a distinct component, the modeler designs *N* simulation models (one per each behavior) but, the final number of simulation models required for the routing process will be *2N + 1*.

The M&S software tool enhances these advantages as follows: *i)* it provides an easy way for modeling routing processes employing a graph abstraction, *ii)* it reduces the simulation modeling times by building automatically all the behavior required to support the routing task, *iii)* it ensures the correctness of RDEVS models because it validates the abstraction prior the formalization, and *iv)* it reduces the possibility of introducing errors during programming because the modeler only needs to codify a reduced set of methods in a few classes.

5 CONCLUSIONS AND FUTURE WORK

The use of the *abstraction-formalization-implementation* approach as foundation for building a M&S software tool leads to a new type of modeling task. In this paper, we present a M&S software tool (developed as a plugin for Eclipse IDE) that employs this approach with aims to build automatically discrete-event simulation models for routing processes thought the graphical definition of abstraction models. Such software tool has been successfully used for the M&S of routing problems related to software engineering and electric power systems fields.

The *abstraction-formalization-implementation* approach used for RDEVS formalism ensures a set of modeling desired properties, such as: *i) Appropriate level of abstraction and separation of concerns that give M&S solutions with low coupling and high cohesion*: The mapping between the abstraction and formalization metamodels provides a separation of the routing process structure and the M&S logic. While the routing process structure is directly mapped to the RDEVS models (that ensures a correct implementation of the routing behavior), the M&S logic is passed to the modelers (with aims to define explicit domain behaviors). This separation of concerns improves the models modifiability because it leads to modular designs with low coupling and high cohesion.; and *ii) Reusability, modifiability and maintainability*: These properties can be analyzed from two different points of view. Given the own definition of RDEVS formalism, the simulation models (in this case, obtained from the transformation process) are easy to reuse and modify. An *essential model* can be reused in several *routing models* and, in the same way, the same *network model* can be used to support distinct routing processes (changing the *routing policies* of the *routing models* already defined). On the other hand, the utility provided by the extension points of the *RDEVS software framework* allows maintaining the implementation of the routing behavior as an isolate software module. Changes performed over this behavior will improve its execution, but will not involve modifications in the specific extensions (that is, the specific RDEVS models designed from the transformation process).

We argue that the M&S software tool developed is more suitable than other types of software tools because it employs a domain abstraction as methodology modeling (providing a natural representation of the problem). This characteristic reduces the knowledge required for building the simulation models attached to a specific routing process definition and, therefore, the modeling tasks could be performed by anyone that knows the problem domain. Of course, M&S experts will be needed for building the component behaviors required to execute the final simulation. Still, with the M&S software tool, the M&S process of routing problems is enhanced.

REFERENCES

ACIMS (Arizona Center for Integrative Modeling and Simulation). 2005. DEVSJAVA. http://acims.asu.edu/software/devsjava/, accessed 23th April 2020.

ACIMS (Arizona Center for Integrative Modeling and Simulation). 2009. DEVS-Suite. http://acims.asu.edu/software/devs-suite/, accessed 23th April 2020.

Bergero, F., and E. Kofman. 2011. “PowerDEVS: A Tool for Hybrid System Modeling and Real-Time Simulation”. *Simulation* 87(2):113–132.

Blas, M. J., S. Gonnet, and H. Leone. 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*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 774-785. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bonaventura, M., G. Wainer, and R. Castro. 2013. “Graphical Modeling and Simulation of Discrete-Event Systems with CD++Builder”. *Simulation* 89(1):4–27.

Capocchi L, J. F. Santucci, B. Poggi, and C.Nicolai. 2011. “DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems”. In *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 27th–29th, Paris, France, 170–175.

Cetinkaya D., A. Verbraeck, and M.D. Seck. 2010. “A Metamodel and a DEVS Implementation for Component Based Hierarchical Simulation Modeling”. In *Proceedings of the Spring Simulation Multiconference*, April 11th-15th, Orlando, United States, 130-137.

Dalle, O., J. Ribault, and J. Himmelspach. 2009. “Design Considerations for M&S Software”. In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 944-955. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hitz, M., and B. Montazeri. 1995. “Measuring Coupling and Cohesion in Object-Oriented Systems”. In *Proceedings of the International Symposium on Applied Corporate Computing*, October 25th-27th, Monterrey, Mexico, pp. 25-27.

Johnson, R. E., and B. Foote. 1988. “Designing Reusable Classes”. *Journal of Object-Oriented Programming* 1(2):22–35.

Kim, S., H. S. Sarjoughian, and V. Elamvazhuthi. 2009. “DEVS-Suite: A Simulator Supporting Visual Experimentation Design and Behavior Monitoring”. In *Proceedings of the Spring Simulation Multiconference*, March 22th-27th, San Diego, United States, 1-7.

Klatt, B., and K. Krogmann. 2008. “Software Extension Mechanisms”. In *Proceedings of the International Workshop on Component-Oriented Programming*, July 7th, Karlsruhe, Germany, 11-18.

Law, A.M. 1991. “Simulation-Models Level of Detail Determines Effectiveness”. *Industrial Engineering* 23(1):16-18.

Madsen, K. 2003. “Five Years of Framework Building: Lessons Learned”. In *Companion of the Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, October 26th-30th,Anaheim,United States,345–352.

Nikolaidou, M., V. Dalakas, L. Mitsi, G. D. Kapos, and D. Anagnostopoulos. 2008. “A SYSML Profile for Classical DEVS Simulators”. In *Proceedings of the International Conference on Software Engineering Advances*, October 26th-31th, Sliema, Malta, 445-450.

Ören, T. I. 2007. “The Importance of a Comprehensive and Integrative View of Modeling and Simulation”. In *Proceedings of the Summer Simulation Multiconference*, July 15th-18th, San Diego, United States, 996-1006.

Page-Jones, M. 1980. *The Practical Guide to Structured Systems Design*. Yourdon Press, New York.

Quesnel G, R. Duboz, E. Ramat, and M. K. Traore. 2007. “VLE: A Multimodeling and Simulation Environment”. In *Proceedings of the Summer Simulation Multiconference*, July 15th-18th, San Diego, United States, 367-374.

Robinson, S. 2008. “Conceptual Modelling for Simulation. Part I: Definition and Requirements”. *Journal of the Operational Research Society* 59(3):278-290.

Robinson, S., R. Brooks, K. Kotiadis, and D. Van Der Zee. 2010. *Conceptual Modeling for Discrete-Event Simulation.* CRC Press.

The Eclipse Foundation. 2019. Acceleo. https://www.eclipse.org/acceleo/., accessed 23th April 2020.

The Eclipse Foundation. 2020a. Eclipse. https://www.eclipse.org/., accessed 23th April 2020.

The Eclipse Foundation. 2020b. Sirius. https://www.eclipse.org/sirius/., accessed 23th April 2020.

The Eclipse Foundation: Eclipse Modeling Project. 2020. Eclipse Modeling Framework. https://www.eclipse.org/modeling/emf/., accessed 23th April 2020.

Touraille, L., M. K. Traoré, and D. R. Hill. 2011. “A Model-Driven Software Environment for Modeling, Simulation and Analysis of Complex Systems”. In *Proceedings of the Spring Simulation Multiconference*, April 4th-7th, Boston,United States, 229-237.

Van Tendeloo, Y., and H. Vangheluwe. 2017. “An Evaluation of DEVS Simulation Tools”. *Simulation* 93(2):103-121.

Wainer, G. 2002. “CD++: A Toolkit to Develop DEVS Models”. *Software Practice and Experience* 32(13):1261–1306.

Wainer, G. 2017. *Discrete-Event Modeling and Simulation: A Practitioner's Approach.* CRC press.

Zeigler, B. P. 2018. “Closure Under Coupling: Concept, Proofs, DEVS Recent Examples”. In *Proceedings of the Spring Simulation Multiconference*, April 15th-18th, Baltimore, United States, 1-6.

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

AUTHOR BIOGRAPHIES

MARIA JULIA BLAS is an assistant professor in the Department of Information Systems at Universidad Tecnológica Nacional (Argentina) and a postdoctoral researcher at Instituto de Desarrollo y Diseño INGAR (Argentina). She holds a PhD in Engineering and Information Systems from Universidad Tecnológica Nacional (Argentina). Her research interest is improving discrete-event modeling and simulation using software enginerring techniques. Her email address is mariajuliablas@santafe-conicet.gov.ar.

SILVIO GONNET received his PhD in Engineering from Universidad Nacional del Litoral (Argentina) in 2003. He currently holds a Researcher position at Instituto de Desarrollo y Diseño INGAR (Argentina). His research interests are models to support design processes and conceptual modeling. His email address is sgonnet@santafe-conicet.gov.ar.