Engineering Science and Technology, an International Journal 24 (2021) 22–34

HOSTED BY

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Full Length Article

# Computer-aided design for building multipurpose routing processes in discrete event simulation models

María Julia Blas *, Silvio Gonnet

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

ARTICLE INFO

*Article history:*
Received 29 July 2020
Revised 30 October 2020
Accepted 2 December 2020

*Keywords:*
Discrete Event System Specification (DEVS)
Computer-aided design
Model-driven engineering
Routed DEVS
Meta-modeling

ABSTRACT

Good domain-modeling enables an appropriate separation of concerns that improves quality properties in the simulation models, such as modifiability and maintainability. In this paper, the interplay of abstraction and concreteness in advancing the theory and practice of Modelling and Simulation is improved using the Model-Driven Engineering levels for building simulation models devoted to routing processes. The definition of this type of processes is detailed as a domain-model conceived as an abstraction defined in a graph model. Such abstraction turns into a set of formal simulation models that are (later) translated into an executable implementation. The final simulation models are specified using Routed DEVS formalism. The methodological proposal is accomplished with the development of a Modelling and Simulation graphical software tool that uses the set of models (defined in terms of the Model-Driven Engineering approach) as the core of its operation. This graphical software tool is developed as a plug-in for Eclipse Integrated Development Environment with aims to take advantage of existent Modeling and Simulation software. Therefore, the usefulness of graphical modeling for supporting the development of the simulation models is empowered with a Model-Driven Engineering process. The main benefit obtained when the Model-Driven Engineering approach is used for modeling an abstraction of the final simulation model is a clear reduction of formalization and implementation times.


## 1. Introduction

In the last years, the Modeling and Simulation (M&S) field has grown becoming a discipline by itself. A good conceptual model lays a strong foundation for successful simulation modeling and analysis [1]. According to Guizzardi and Wagner [2] the terms 'domain model' and 'conceptual model' are equivalent. In this sense, conceptual modeling serves as a bridge between the problem owner and the simulation modeler.

Frequently, simulation models attempt to solve situations where a specific set of components send/receive interactions (e.g. messages or events) employing a selective mechanism. These situations are commonly found in domain areas such as telecommunication, supply chain, business process management, mobile applications, social networks, and so on. For well-defined mechanisms (for example, publish-subscribe protocol [3]) several well-established implementations exist. However, when the selective mechanism is defined as part of the situation description, new modeling strategies should be applied. In these cases, the modeler designs the simulation model as a combination between its domain components and the selective mechanism description (i.e., a routing process description).

The Discrete-Event System Specification (DEVS) formalism [4] is a modeling formalism based on systems theory that provides a general methodology for the hierarchical construction of reusable models in a modular way. Over the years, DEVS has found an increasing acceptance in the model-based simulation research community becoming one of the preferred paradigms to conduct modeling and simulation enquiries [5]. When DEVS is applied for modeling situations that involve their own selective mechanism, DEVS-modelers use one of the following strategies: *i)* "pre-wired" connections (i.e., an implicit routing formulation), or *ii)* "handlers" (i.e., an explicit simulation component that manages the routing functionality). One way or another, the modeler solves the situation using its knowledge regarding DEVS formalism. Therefore, its design is based on the formalism itself (not in the domain study). This is because, by nature, simulation is a technical field [6].

However, the design of the simulation model impacts all aspects of the study, in particular the data requirements, the speed

* Corresponding author.
*E-mail address:* mariajuliablas@santafe-conicet.gov.ar (M.J. Blas).
Peer review under responsibility of Karabuk University.

https://doi.org/10.1016/j.jestch.2020.12.006
2215-0986/

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 [7]. Then, conceptual modeling can be used as a guideline when building DEVS simulation models that involve routing functionality.

According to Robinson [8], conceptual modeling is about moving from a problem situation, through model requirements to a definition of what is going to be modeled and how. Moreover, from the modeling perspective, a simulation model is similar to a software system model because both kinds of models can be developed from a conceptual system model [9]. In Software Engineering (SE), this approach is called Model-Driven Engineering (MDE). MDE 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 [10]. However, in the M&S field, there is no common understanding on how to apply these modeling levels when building simulation models.

Given that MDE produces well-structured and maintainable systems and increases the level of abstraction [11], it is our research goal to use MDE as a mechanism for improving the M&S of routing processes. Several authors have studied different types of processes using DEVS (or variants of DEVS) as support M&S mechanism. Frequently, these researches are based on domain-models and Model-Driven approaches. For example, Zacharewicz et al. [12] propose a description language for workflow processes along with an automatic transformation of a workflow into a Generalized-DEVS model. More recently, in the context of Business Process Models, a Model-Driven Development framework for the M&S has been presented [13]. Such a framework is based on model-to-model transformations from BPMN (as a conceptual modeling language) to DEVS (as a simulation model specification). In this direction, Bazoun et al. [14] have proposed a refinement of the previous BPMN metamodel with new concepts and transformation rules to adapt the existing model to BPMN 2.0. In this paper, we abstract the routing process definition into a graph model that is used as a domain model description for building (automatically) a discrete-event simulation model of the original situation without requiring any interaction with the DEVS-modeler. Instead of using a modeling language (e.g., UML or BPMN), we propose a simple graph representation based on nodes and links to describe the structure of our processes. To improve the performance of the final simulation models, we employ the Routed DEVS (RDEVS) definition [15] as a "layer" above DEVS with aims to provide routing functionality without requiring the user to "dip down" to DEVS itself for any functions [16]. The conceptualizations developed using the MDE approach are encapsulated in a M&S software tool implemented as a plug-in for Eclipse Integrated Development Environment (IDE) [17]. Our aim is *i)* to offer a software environment for the graphical modeling of routing situations using a standardized graph description, and *ii)* to be able to generate 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 structured as follows. Section II briefly introduces the different uses of MDE in the M&S field. It also describes the principles of DEVS formalism and how DEVS models can be used for solving routing processes. A description of the desirable capabilities of M&S software tools is also included. Section III presents the core of the proposal: the MDE modeling levels used as a conceptual view of routing situations. It describes the metamodels defined as conceptualizations of each MDE level to represent some view of the original situation. Section IV introduces the M&S software tool designed as a plug-in for Eclipse IDE using the metamodel obtained from the MDE approach. The goal of this plug-in is to provide a single object-oriented M&S tool for building simulation models of routing situations from graph models. Section V details the main benefits of having the M&S tool with a proof of concepts. Finally, Section VI is devoted to conclusions and future work.

## 2. Related work

### 2.1. Model-driven-X in the M&S field

The Model-Driven-X (MDX) methodologies are widely used in the SE field. The Model-Driven Development (MDD) proposes a paradigm that uses models as the primary artifacts and redefines the implementation as automatic generation from the models [18]. Model-Driven Architecture (MDA) is a particular vision of MDD proposed by the Object Management Group [19]. The MDE approach follows the MDA proposal to cover all engineering process areas with a focus on developing metamodels to facilitate automated transformations.

The M&S field has used MDX approaches in different ways. For example, Alshareef et al. [20] propose the use of MDE in simulation especially for creating platform-independent models. Other authors believe that MDE can be used for modeling conceptual views before building the final simulation model [21,22]. In this paper, we employ MDE as a vehicle for designing and implementing a M&S software tool that provides DEVS-based solutions for generic routing processes. We argue that MDE modeling levels can be used to support the abstraction, formalization, and implementation meta-models required to obtain (automatically) the final simulation model.

However, it is important to remark that simulation establishes a model in a computational environment. Such a "computation" is an aspect of M&S and does not imply that M&S is a subfield of SE [23]. Even when MDX approaches have been used in M&S, Model-Based (MB) paradigms (such as MB System Engineering and MB Simulation) are frequently used to get solutions to M&S issues [24,25]. Such MB approaches use MDX practices pragmatically (i.e., the models are important, but they do not necessarily drive the development process).

### 2.2. Discrete event system specification

The DEVS formalism is a complete M&S technique based on systems theory introduced by Prof. Zeigler in the early 70's. It provides a general methodology for the hierarchical construction of reusable models in a modular way. It employs two abstraction levels to describe the behavior of a system: atomic and coupled.

At the atomic level, the behavior of the system is detailed as a sequence of deterministic transitions among sequential states. This sequence determines how the system reacts to external events. Also, it describes how output events are created.

Equation (1) resumes the atomic model definition following the approach called "Classic DEVS with ports" [4].

$$M = (X, Y, S, \delta ext, \delta int, \lambda, ta) \quad (1)$$

where:

$X = \{(p,v) \mid p \in InPorts, v \in X_p\}$ is the set of input ports and values;
$Y = \{(p,v) \mid p \in OutPorts, v \in Y_p\}$ is the set of output ports and values;
S is a set of sequential states;
$\delta ext$: $Q \times X \rightarrow S$ is the external transition function, where
$Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the total state set in which e is the elapsed time since the last transition in the state s;
$\delta int$:$S \rightarrow S$ is the internal transition function,
$\lambda$: $S \rightarrow Y$ is the output function; and

ta: $S \rightarrow R_0^+$ is the advance time function.

On the other hand, at the coupled level, the system is described as a network of interconnected components. Such components can be atomic or coupled models in their own right. The connections among components (i.e., the couplings) denote how components influence each other. There are three types of influences: *i)* external input couplings that connect external inputs to component inputs, *ii)* external output couplings that connect component outputs to external outputs, and *iii)* internal couplings that connect component outputs to component inputs.

Equation (2) summarizes the coupled model definition using, also, the "Classic DEVS with ports" approach.

$$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, Select) \quad (2)$$

where:

X and Y are defined in the same way as Eq. (1);
D is the set of the component names;
For each $d \in D$, $M_d$ is a DEVS model;
$EIC \in \{((N,ip_N),(d,ip_d)) \mid ip_N \in InPorts, d \in D, ip_d \in InPorts_d\}$ is the set of the external input couplings;
$EOC \in \{((d,op_d),(N,op_N)) \mid op_N \in OutPorts, d \in D, op_d \in OutPorts_d\}$ is the set of external output couplings;
$IC \in \{((a, op_a),(b, ip_b)) \mid a,b \in D, op_a \in OutPorts_a, ip_b \in InPorts_b\}$ is the set of internal couplings; and
Select is the tie-break function.

### 2.3. Structuring routing processes with DEVS: Routed DEVS

When solving a routing process situation with DEVS, the final model employs one of the following strategies: *i)* "pre-wired" connections (i.e., an implicit routing formulation), or *ii)* "handlers" (i.e., an explicit component that manages the routing functionality). In *i)*, the simulation models are designed considering the events flow as pre-defined connections. For each possible interaction among components that defines an alternative flow, the model includes a coupling among ports to be used when the interaction is active [14,26–29]. On the other hand, the solutions designed according to *ii)* use a set of simulation components (often called "handlers") that are explicitly introduced in the simulation model with aims to manage the routing functionality [30,31].

To overcome the implicit and explicit routing functionality, the RDEVS formalism provides an appropriate separation of concerns in terms of the routing structure, the possible paths, and the component behaviors. Over the years, researchers have proposed extensions of DEVS with aims to solve different situations. In this context, the RDEVS formalism was designed as a DEVS extension that improves the modeling of DEVS simulation models devoted to routing process situations [15]. Unlike DEVS formalism, the RDEVS formalism defines three types of simulation models: *essential*, *routing*, and *network*.

The *RDEVS essential model* specifies a discrete-event simulation model that exhibits the behavior of an elemental domain-component involved in the routing process situation. The guidelines for building an *essential model* should be defined by a domain expert. Formally, an *RDEVS essential model* is defined as a DEVS atomic model [4].

The *RDEVS routing model* defines a discrete-event simulation model that acts inside the routing process situation. Its definition employs a domain-component as an operational description of its own behavior. Hence, the *routing model* definition embeds an instance of a specific *essential model*. Therefore, the same *essential model* can be embedded in several *routing models*. Besides the domain-component definition, the *routing model* requires a *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*. The *routing policy* definition includes an identifier used to distinguish the set of routing models that build the routing situation. 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 situation 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 creates 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 composes the routing policy. Therefore, the execution of the routing process is build-in inside the RDEVS models.

Finally, the *RDEVS network model* describes a complex discrete-event simulation model that solves a routing situation as a routing process. 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 leave the routing task to the *routing policies* (detailed in each *routing model*). The *network model* specification also involves two special translation functions (used to link multiple networks): *input translation function* and *output translation function*. These functions allow matching *events with identification* produced by different routing processes (i.e., different *network models*). 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.

Then, the RDEVS formalism provides a feasible solution to modelers for building discrete-event simulation models for routing processes. To show the dependencies among RDEVS models, Fig. 1 presents two examples ($ROUTING_{SITUATION\ 1}$ and $ROUTING_{SITUATION\ 2}$) based on the same set of domain-components ($MACHINE_{TYPE\ A}$ and $MACHINE_{TYPE\ B}$). In both examples, the basic domain-components are defined as *essential models* (that is, DEVS atomic models). Over these models, a set of *routing models* is defined with aims to represent the components to be used in each situation. Each *routing model* embeds an essential model with aims to define its behavior. The routing process situations are modeled as individual *network models* structured with these *routing models*.

Then, the RDEVS formalism allows modeling different routing process situations using the same set of DEVS models (defined as

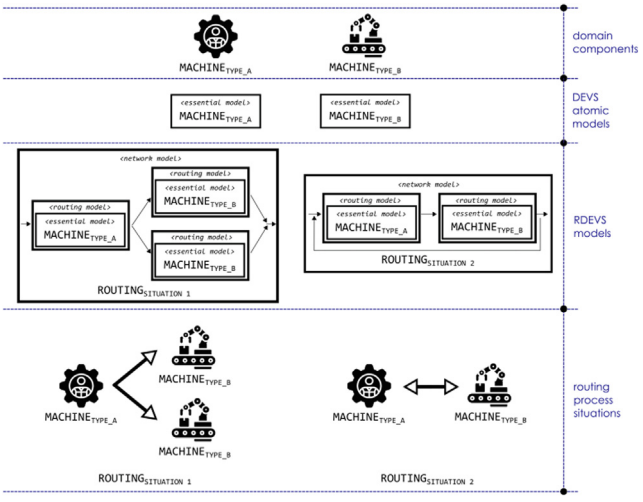


**Fig. 1.** Dependencies among DEVS and RDEVS models.

*RDEVS essential models*) for domain-components. That is, the RDEVS models act as a "layer" above DEVS with aims to provide routing functionality without requiring "dip down" to DEVS itself for any functions [16]. Since RDEVS is a subclass of DEVS [32], the RDEVS models can be executed using DEVS simulators.

### 2.4. Capabilities of the M&S software products

Well-designed M&S software products should support the reuse of existing (software) components [33]. 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. Also, cohesion is an important attribute regarding the quality of software modules. Good abstractions typically exhibit high cohesion [34]. Thereby, software modules defined in an M&S software product should fully capture explicit abstractions of the M&S task.

Hence, any M&S software tool should (at least) provide two set of modules: software modules for supporting the modeling, and software modules for supporting the simulation execution. Moreover, these products should provide not only simulation capabilities but, also, they should support new modeling strategies that improve the modeling task (e.g. number of simulation models to be developed, time to spend for getting correct implementations, modeling complexity, M&S knowledge required prior to perform the modeling, etc.).

Nowadays, there are multiple software tools and simulators for DEVS models. Van Tendeloo and Vangheluwe [35] discuss the main functionalities of the most frequently used M&S tools. As the authors describe, there is a set of DEVS M&S tools that support graphical modeling capabilities. For example, PowerDEVS [36] 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 [37] can be used to create models for CD++ [38]. DEVSimPy [39] offers a graphical modeling environment for coupled models. A similar approach is applied in Virtual Laboratory Environment [40] where atomic models are written in C++ and coupled models can be created using either the graphical environment or by manually writing XML files.

Even when existent capabilities regarding graphical modeling are useful for DEVS modelers, these approaches are centered on predefined formalizations of the simulation model. That is, all the approaches are centered on building graphically DEVS models that have been already designed (at least, at the conceptual level) by the modeler. Then, the modeler (that already knows DEVS formalization) employs such knowledge in the definition of the simulation models. In this sense, the M&S field has applied the MDE approach in several cases. For example, De Lara and Vangheluwe [41] present a tool for metamodeling and model transformations for the simulation that facilitates computer-assisted modeling of complex systems. Other approaches related to MDE are described by Levytskyy et al. [42] and D'Ambrogio et al. [43]. Still, these approaches are not focused on modeling the original problem or situation. From this perspective, the novel contribution of this paper is the way in which the MDE approach is used in the M&S field. The MDE modeling levels are used as a guideline in the model definition through the abstraction, formalization, and implementation of the original situation. Then, our M&S tool uses a graphical representation of the original routing situation as a marker for building (automatically) a DEVS implementation of the simulation models. This is the main difference between the M&S software tool developed in this paper and the existing ones.

## 3. MDE modeling levels for M&S of routing situations

MDE differences three types of models as engineering artifacts resulting from the activities performed in the analysis, design, and implementation phases. These models are defined as:

1. Domain models: Solution-independent descriptions of a domain produced in the analysis phase of a SE project.
2. Platform-independent design models: General computational solution developed on the basis of the domain model at the design phase of the SE project.
3. Platform-specific implementation models: The implementation model is encoded in the programming language of the target platform at the implementation phase of the software project.

Then, a software system model consists of a set of models defined in terms of the most important viewpoints of the system to crosscut all three modeling levels [2].

From the modeling perspective, a simulation model is similar to a software system model. Following a scientific engineering approach, Guizzardi and Wagner [9] claim that both kinds of models are derived from a conceptual system model (also called *domain model*). The main benefit obtained from the definition of a *domain model* is the clarification of the real-world semantics. In SE, this approach is based on MDE. In this case, a clearly defined semantics of the conceptual model of a domain leads to a higher overall quality of the software application system built upon that model with respect to comprehensibility, maintainability, interoperability, and evolvability [9]. Since simulation models are more about the real world than software system models, the principles of MDE are useful. In fact, in the M&S field, such a *domain model* is often called an *abstraction model*. An *abstraction model* focuses on an aspect of reality and, almost by definition, greatly reduces the complexity of the reality being considered [4]. Then, it focuses on the perspective and language of the subject matter experts for the domain under consideration.

Building a *domain model* that works in any routing situation is impossible. However, an appropriate abstraction can be designed with aims to fulfill most routing situations. This is the approach described in Section 3.1 for getting the metamodel of the domain model.

Once the first MDE modeling level is defined, a *formal model* of the domain (model) is required. *Formal models* generally describe the structure and the behavior of the system independent from the implementation details [11]. Commonly, a *formal model* makes it easier to work out implications of the defined *abstraction* with aims to get a future *implementation*. If multiple formalisms are used to support the model definition, the same *abstraction model* can be used to define several *formal models*. Hence, a formal model follows the guidelines of a *platform-independent design model* (where the same *domain model* can potentially be used to produce different *design models*). Section 3.2 describes how the *domain model* of Section 3.1 is formally defined as a simulation model in RDEVS. In this case, RDEVS is used as the formalization mechanism in a *platform-independent* way.

Finally, once the second MDE modeling level is obtained, an *implementation model* can be defined with aims to get a concrete realization of the formalization. Commonly, *implementation models* are deployed in a simulation environment. Therefore, this model can be seen as a *platform-specific implementation model* (third MDE modeling level). Section 3.3 presents the RDEVS *implementation*

*model* using Java as the target-platform for the simulation execution.

### 3.1. Routing process situations as graph models

The success of the modeling task depends on the modeler's ability to get an appropriate level of abstraction in the design. In this context, graph models (defined as a set of nodes and edges among the nodes) can be used as an abstraction of routing situations (reality). Then, a routing situation description can be seen as a graph model.

When routing situations are modeled as graphs, 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, domain-components are conceptualized as nodes of the graph that share a behavioral operation. That is, several nodes are used to represent the same domain-component.

Fig. 2 shows a UML class diagram that describes the 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* (multiplicity 1). Finally, a *Node Instantiates* a *Component* that has 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...*).

The *abstraction metamodel* (Fig. 2) 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 an abstraction of routing situations is that existent metrics designed for such models can be used as support measures for studying the complexity of the simulation models obtained from it.

### 3.2. Graph models as routed DEVS models

Formalization makes it easier to work out the implications of the abstraction and implementation in reality [4]. As Section 2.2 details, the RDEVS formalism is designed to level out the modeling effort of routing situations modeled in DEVS. Hence, the RDEVS formalism can be used as formalization language for the abstraction model depicted in Fig. 2.

Following this approach, each RDEVS model can be seen as the formalization of some component defined in Fig. 2 as follows: *i)* the *essential model* formalizes a *component* (because the *essential model* is designed to exhibit the behavior of some domain-component), *ii)* the *routing model* formalizes a *node* of the *graph* (because the *routing model* is designed as part of the routing situation), and *iii)* the *network model* formalizes the *graph* (because the *network model* is designed to represent the overall routing situation).

Fig. 3 extends the UML class diagram depicted in Fig. 2 with the *formalization metamodel* (i.e., concepts highlighted in yellow)

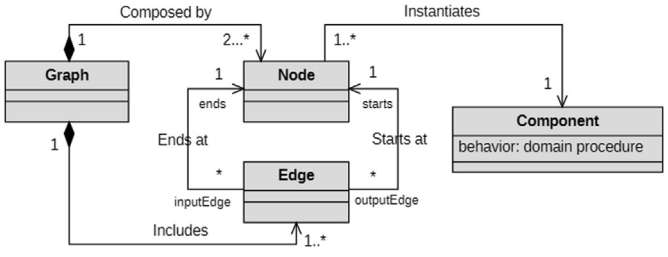


**Fig. 2.** Simplified UML class diagram that depicts the main concepts and relationships required for instantiating a graph model that represents a routing situation (*abstraction metamodel*).

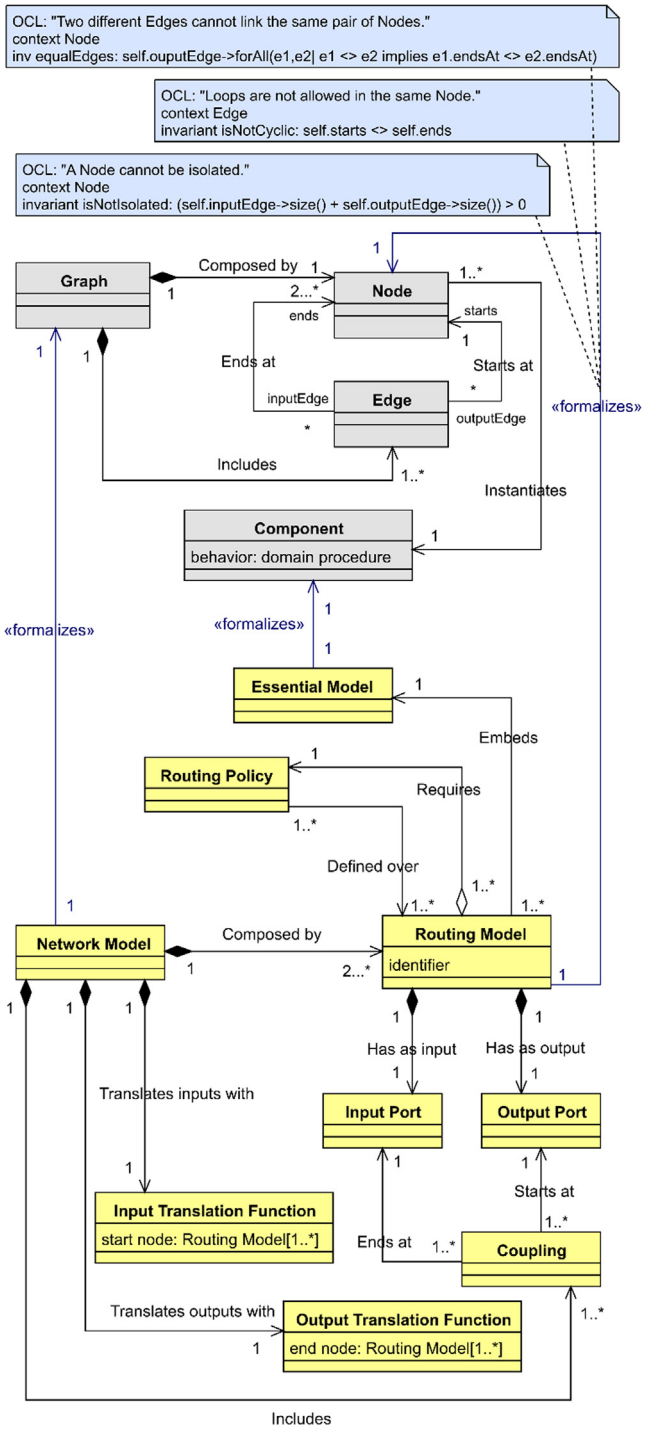


**Fig. 3.** UML class diagram that links the abstraction metamodel with the RDEVS specification through the *formalizes* relation. The associations that hold the same name and multiplicity in both metamodels describe the same dependency between elements. The *Embeds* association (from the *formalization metamodel*) is the materialization of the *Instantiates* association (from the *abstraction metamodel*). The OCL constraints are used to ensure the correct design of RDEVS models.

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 Functions* 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 to maintain consistency between *abstraction* and *formalization metamodels*. This consistency is related to the correctness of the formalism. Even when RDEVS models can be used to support the design of routing situations, the graph model used as *abstraction* must ensure a set of structural properties prior its *formalization* with RDEVS. Such structural properties guarantee the validity of the RDEVS models. In this context, the OCL constraints detailed in Fig. 3 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 before 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* must be connected to each other). Then, before obtaining such a simulation model, the OCL invariant named *isNotIsolated* should be evaluated.

As a remark, from a conceptual point of view, a *Coupling* can be seen as a formalization of an *Edge*. However, Fig. 3 does not include such formalization as a relationship. This is because the *Couplings* in RDEVS models are structured by the *Network Model* as all-to-all connections. When a *Routing Model Embeds* an *Essential Model*, the *Routing Policy* should 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*) 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...*). Hence, the *Edges* are used as guidelines for building *Routing Policies* but not as abstraction of the *Couplings*.

Now, the definition of RDEVS simulation models over the graph provides an appropriate separation of concerns between the routing situation (i.e., the structure and routing paths) and domain-components in order to get a full simulation model from the original scenario. An important remark is that there is no universal routing policy to manage the global structure of the situation. Each *routing model* takes its own decisions (through the routing policy) on how to direct the input and output events. The advantage is that the modeler does not need to explicitly define any additional information to manage the routing during the design phase. All the information required for building the routing policies is defined in the structure of the original routing situation.

### 3.3. Routed DEVS models as discrete event models implementation

To get a concrete realization of the formalization defined in Section 3.2, such representation should be implementable in a programming language. Once an encoded version of the RDEVS models defined from the formalization is instantiated, such implementation can be seen as a concrete realization (e.g. a simulation model encoded in Java) of the abstraction (i.e., the routing situation) through the formalization defined (i.e., RDEVS models).

In addition to the reusable functions provided in libraries, software frameworks provide "flows of control" [44]. Frameworks shall ease/speed up the development of software from the domain they are created for [45]. Therefore, a software framework may be built on top of a set of libraries and might be used to create more specialized solutions. Since RDEVS formalism is designed as a subclass of DEVS [32] and RDEVS models can be executed using a DEVS simulator, the RDEVS models can be implemented as extensions of DEVS models. In this context, available libraries of DEVS formalism can be used as a guideline to develop RDEVS implementations.

From this perspective, a software framework for building RDEVS models was developed using DEVSJAVA library [46] as an underlying M&S layer. DEVSJAVA is a M&S tool implemented in Java that supports characterizing models in DEVS formalism. Through the extension of DEVSJAVA, the RDEVS software framework provides a solid solution for building executable models defined in RDEVS formalism. Moreover, the DEVSJAVA Viewer can be used to get a visual representation of RDEVS models.

The RDEVS framework was initially presented by Blas et al. [15]. The main classes included in such a framework were the following:

- EssentialModel.java, RoutingModel.java, and NetworkModel.java to define RDEVS simulation models,
- RoutingFunction.java and RoutingFunctionElement.java to define the Routing Policy used as part of the Routing Model definition,
- InputTranslationFunction.java to define the transformation from events to events with identification, and
- OutputTranslationFunction.java to define the transformation from events with identification to events.

However, new classes were added to improve the framework definition. For example, the class *IdentifiedMessaje.java* was added with aims to define the structure of events managed by the simulation models as part of the routing functionality. All the classes developed as part of the framework were designed to depict concepts related to RDEVS formalism. Hence, the RDEVS software framework enhances the development of RDEVS simulation models in Java programming language using some features provided by DEVSJAVA.

Fig. 4 depicts the classes implemented in the final version of the RDEVS framework. The relationships are used to represent dependencies among classes. The classes highlighted in blue belong to DEVSJAVA. These Java classes can be seen as potential extension points of the RDEVS software framework for building RDEVS implementations. In SE, an extension point is the definition of the provided interface for extensions [47]. That is, an extension itself is an implementation according to an extension point (equal to an implementation of a software component). Therefore, the RDEVS software framework includes extension points configured for designing explicit instances of RDEVS models as reusable software components slated for executing the routing simulation (already defined in the framework) without any other consideration.

Hence, each concept defined in the *formalization metamodel* is implementable over an extension point of the framework (i.e., a new Java class based on the existent ones). Fig. 5 extends the *abstraction-formalization metamodel* using the *implementable* stereotyped relationship (association highlighted in red) to show the dependencies between RDEVS formal models and their Java implementations. The classes with orange background belong to the *implementation metamodel*. Then, Fig. 5 defines the *abstraction-formalization-implementation metamodel* for routing situations. Now, when a routing situation is modeled as a *Graph*, the formal definition of the simulation model is a *Network Model* implementable as an extension of the Java class named *NetworkModel.java*.

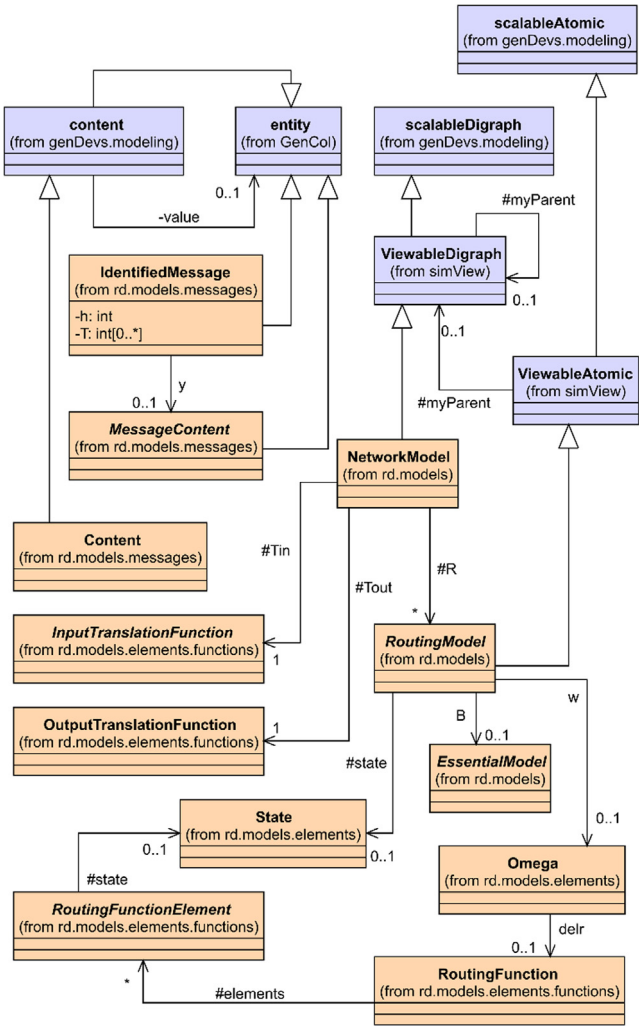


**Fig. 4.** Java classes implemented as part of the RDEVS framework (using DEVSJAVA as a support mechanism).

## 4. Graphical software tool for the M&S of routing situations

Graphical modeling is an aspect that is used successfully [48]. The advantages of a graphical M&S software tool are:

1. *easy-to-use*: in graphical modeling software, the modeling is performed by manipulating graphical elements and their connections;
2. *fast modeling solutions*: graphical modeling software allows the development and solution of complex simulation models rapidly with limited M&S background;
3. *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
4. *standardized simulation models*: through the employment of a set of well-defined graphical components, the graphical modeling software ensures standardized designs [37,49–51].

Then, with aims to build a M&S software tool that provides a full solution for building simulation models for routing situations, a graphical M&S plug-in for Eclipse IDE was developed. The core of this plug-in is the metamodel depicted in Fig. 5.

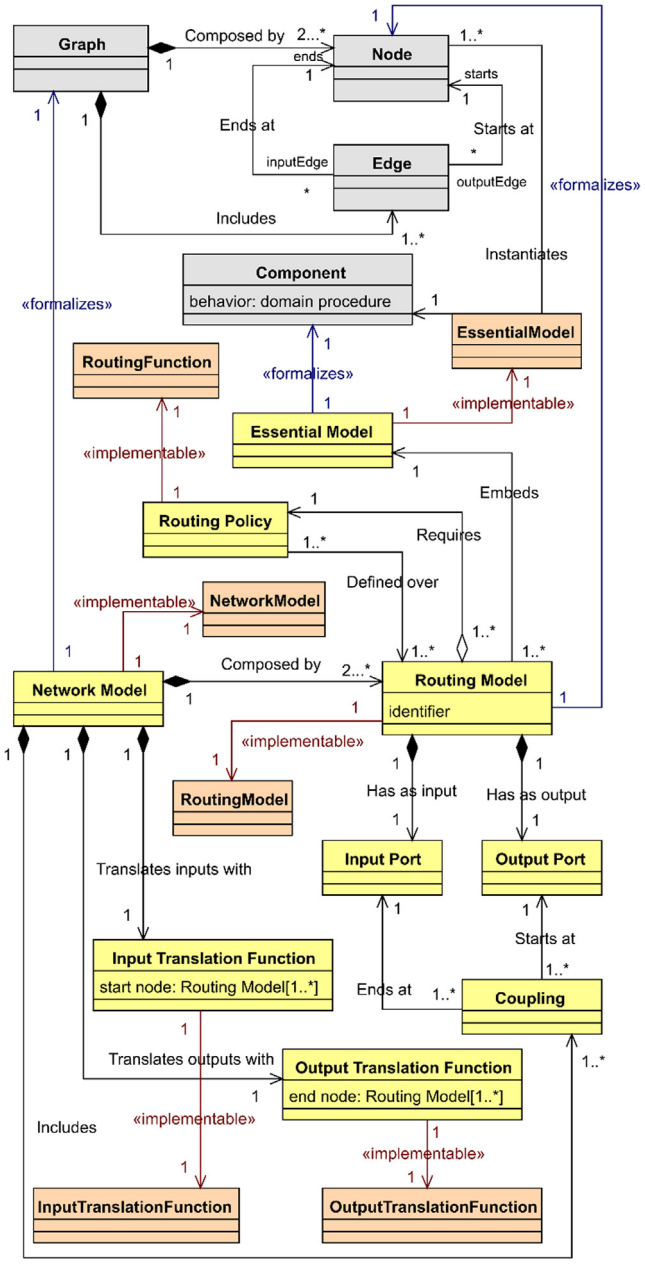


**Fig. 5.** UML class diagram of the *abstraction-formalization-implementation metamodel.*

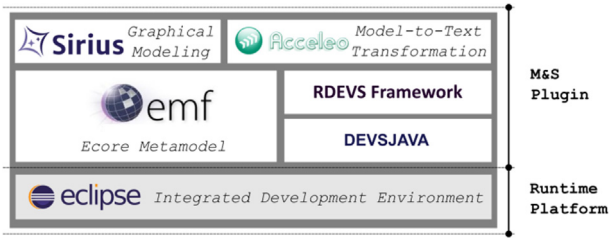


**Fig. 6.** Layered software architecture of the M&S plug-in.

Fig. 6 shows the plug-in architecture as a layered design pattern that is built with five software modules. The *code runtime platform* is Eclipse IDE. The layered architecture used to structure the M&S software allows us to get reusable software modules and, at the

same time, modularity in each level. Besides employing DEVSJAVA and the RDEVS software framework, the M&S plug-in uses other software modules for building the routing representation (i.e., the abstraction, formalization, and implementation metamodels). With aims to ensure full compatibility with the underlying platform, the development of the M&S software employed several plug-ins of Eclipse IDE. Hence, the final M&S graphical tool takes the advantages of Eclipse plug-ins for supporting the development of each module.

The Eclipse Modeling Framework (EMF) [52] was used for building the foundational metamodel (i.e., the *abstraction metamodel*) required as a support mechanism of the plug-in. The Sirius project [53] was employed for developing a graphical representation of the foundational metamodel. Such graphical representation allows building a graphical representation of routing situations (that is, *abstraction metamodel instances* or, simply, *abstraction models*). Such representation is based in a set of graphical elements that depict the concepts and relationships detailed in the metamodel. Finally, the Acceleo development tool [54] was used for translating the *abstraction model* to the *formalization* required to get the *implementation* attached to it.

### 4.1. Ecore metamodel

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 [52].

The foundational metamodel used for modeling the routing process was implemented with EMF to get a data model specification described in XMI. This implementation provides the Ecore metamodel required for building other software modules that compose the M&S plugin. The core of such metamodel was the abstraction metamodel (Fig. 2). 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* with aims to remove the notion of "instance" from the modeling task. Also, new elements were added, including:

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

This last concept (*Abstraction Description* concept) added to the metamodel along with 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 constraints over the metamodels, the OCL constraints attached to the *abstraction-formalization* dependency were also included over the foundational metamodel. With this addition, the *abstraction model* (obtained as an instance of the foundational metamodel) can be validated before its formalization. Then, an explicit and valid abstraction of a routing situation based on the graph model can be instantiated from the Ecore model defined with EMF.

### 4.2. Sirius graphical definition

Sirius is an Eclipse project which allows easily creating graphical modeling workbench tools by leveraging the Eclipse Modeling technologies [53]. Hence, Sirius was employed to define graphical representations for each element included in the Ecore metamodel. A representation was detailed for each concept including:

- graphical icons for concepts *Graph*, *Node*, and *Component*, and
- graphical links for concept *Edge* and *Materializes* association.

Besides providing a graphical instantiation of the Ecore metamodel, Sirius provides validation functionality. A validation button was added to the graphical description with aims to verify the correctness of the *abstraction model* (i.e., the OCL constraints along with multiplicities and attributes uniqueness). If problems are detected, the M&S tool shows an error message with a warning icon over the graphical element. If the model is correct, the modeler can go directly to the *formalization-implementation translation* using the *abstraction model* instantiated.

### 4.3. Acceleo model-to-text transformation

Acceleo is a template-based technology including authoring tools to create custom code generators. It allows automatically producing any kind of source code from any data source available in EMF format [54].

By defining a generation model for the model-to-text transformation, the elements graphically defined in the *abstraction model* are navigated to get their formalizations. These formalizations are used to get the Java code required for the simulation model implementations. Therefore, the *formalization model* attached to the elements defined in the *abstraction model* is obtained as follows:

- each *Component* included in the *Abstraction Description* is formalized in an essential model structure;
- each *Node* included in the *Graph* is formalized in a routing model that encompasses the model identifier (that is defined as a 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*.

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

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

This template shows how the elements included in the abstraction model are navigated (using 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*.

Once the *abstraction model* is fully translated to its equivalent *implementation model*, the modeler needs to specify the only thing missing in the simulation model: the behavior of the components.

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

| Ecore Model | | Java Implementation | |
|---|---|---|---|
| *Concept* | *Prop.* | *New Class (.java)* | *"extends"*[a] |
| Component | name | <name > EssentialModel | Essential Model |
| | | <name > State | State |
| Node | name | <name > RoutingModel | Routing Model |
| | | <name > FunctionElement | Routing Function |
| | | <name > RoutingFunctionElement | Routing Function Element |
| Graph | name | <name > NetworkModel | Network Model |
| | | <name > InputTranslationFunction | Input Translation Function |
| | | <name > OutputTranslationFunction | Output Translation Function |

[a] From the Java classes included in the RDEVS software framework.

Such behaviors are presented to the modeler as TO-DO tasks in the classes named as < *name* > *EssentialModel.java*. It is important to remark that this remaining codification is intended to define the explicit behavior of the components. The missing implementations should be well-known by modelers because they belong to domain-components. Then, this activity should not consume a lot of time.

When such codification is done, the RDEVS simulation model can be executed. The overall behavior required for supporting the routing situation execution is provided by the definition of RDEVS models. Hence, using the M&S software tool, modeling and implementation times are reduced to the automatic translation of an *abstraction model* defined graphically from the original routing situation.

## 5. Benefits of M&S graphical software tool: Proof of concepts

Either with DEVS or RDEVS, routing situations can be modeled as discrete-event simulation models. When RDEVS formalism is used for building such simulation models, the modeler can focus its attention on the domain properties without worry about the routing implementation.

By embedding essential models into the routing models, the RDEVS formalism improves the design phase in two distinct ways:

1. behavioral descriptions are only required for the domain-components (the routing behavior is build-in as part of the RDEVS models),
2. routing policies are isolated from behavioral descriptions (allowing the reuse of essential models in other networks).

Therefore, the RDEVS models allow the modeler to explicitly define the routing functionality without specifying new routing behavioral descriptions. Such routing information is detailed inside the routing model as part of its own definition to authenticate senders and receivers prior executing the (linked) essential model. Then, the RDEVS formalism is designed for level out the modeling effort of routing situations in DEVS by providing an easier modeling solution. Such solution employs a set of models defined in terms of the main elements involved in routing processes (i.e., well-known domain components). Main benefits of using a RDEVS solution against a DEVS solution are discussed by Blas et al. [15].

The RDEVS models built by the modeler in a RDEVS solution are centered on using a good conceptual abstraction of the routing situation. However, even when the formalism embeds the routing functionality, the modeler still needs RDEVS skills for the definition and codification of the final simulation models.

The M&S graphical software tool proposed in this paper enhances these advantages as follows:

1. it provides an easy way for modeling routing situations employing a graph abstraction,
2. it reduces the simulation modeling times by building automatically all the behavior required to support the routing functionality,
3. it ensures the correctness of RDEVS models because it validates the abstraction before the formalization, and
4. 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.

As proof of concepts of these benefits, the process depicted in Fig. 7 is presented as a routing situation. This process is based on the set of domain-components detailed in Fig. 1. Each type of machine is in charge of performing some processing over its inputs. The entire situation takes an empty box and transforms it into a package. However, during the first stage of processing (initial instance of Machine Type A), two possible output routes can be used. In this case, both routes produce the same final result: a package.

With aims to get the discrete-event simulation models linked to this routing situation, the example was modeled in the M&S software tool. Fig. 8 presents a screenshot of such an abstraction model. Following the example, the abstraction model is composed of two *Components* (named "Machine Type A" and "Machine Type B") and five *Nodes* (named "Machine1" to "Machine5"). The *Nodes* are included in a *Graph* named "Example". Each *Node materializes* some *Component* (e.g. the *Node* named "Machine1" materializes the *Component* named "Machine Type A"). Finally, the *Nodes* are linked by *Edges* (grey arrows). These *Edges* depict the event flows allowed in the routes.

With the abstraction model defined, the modeler only has to execute the transformation process to get the implementation of the RDEVS simulation models. This option is available in the M&S tool over the abstraction model (Fig. 9(a)). As a result of such a transformation process, the M&S tool creates a new Java project named as the abstraction model that ends with ".rdevsimplementation" (Fig. 9(b)). This new Java project includes all the Java classes described in Table 1 as part of the "rdevsmodels" package. For example, the class "GraphExample.java" implements the network model that performs the simulation of the "Example" routing situation (abstracted into a Graph). The project also includes the RDEVS framework (the file named "rdevs.implementation.jar"). The code to be implemented by the modeler to detail the behavior of domain-components is shown to the developer as TO-DO tasks (see, for example, lines 21 and 31 in Fig. 9(b)). The entire translation process is performed in a few seconds (in this case, 2.003 s).

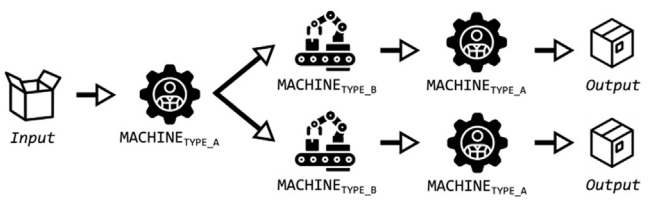


**Fig. 7.** Routing situation (example).

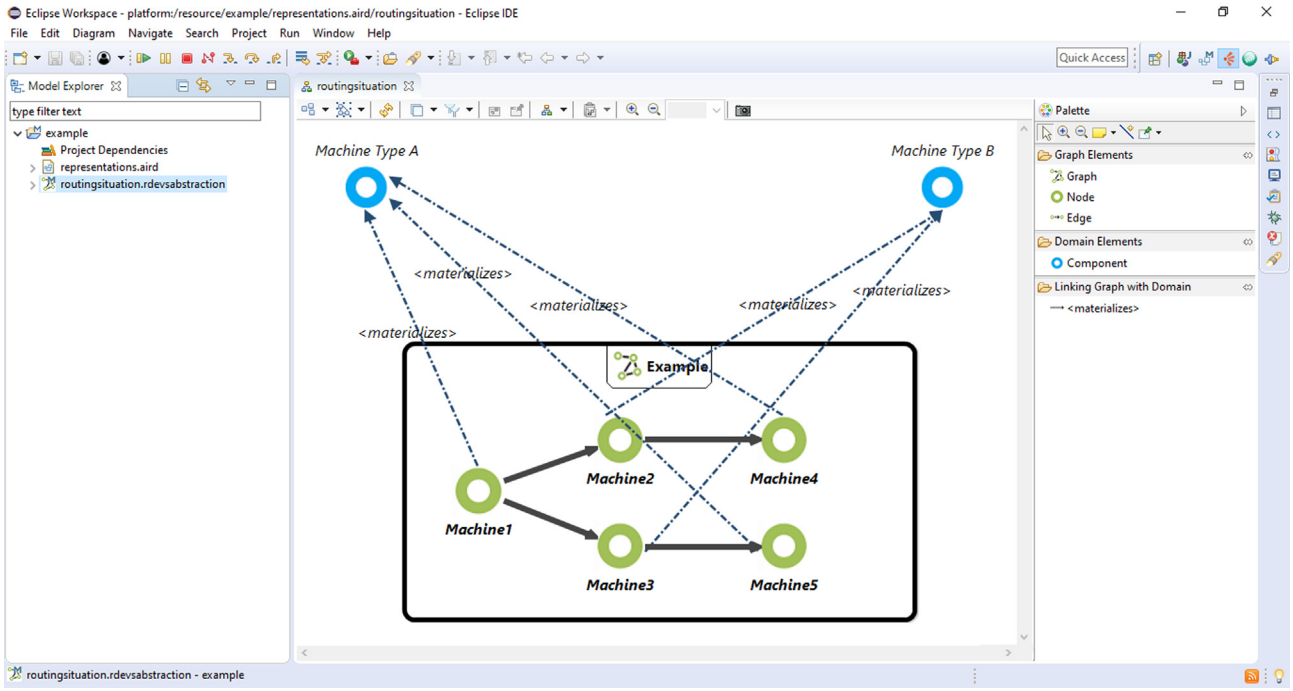


**Fig. 8.** Screenshot of the example modeled with the M&S software tool. The modeler drags and drop the elements available in the palette (right side of the screen) to design the abstraction model (file extension "*.rdevsabstraction").

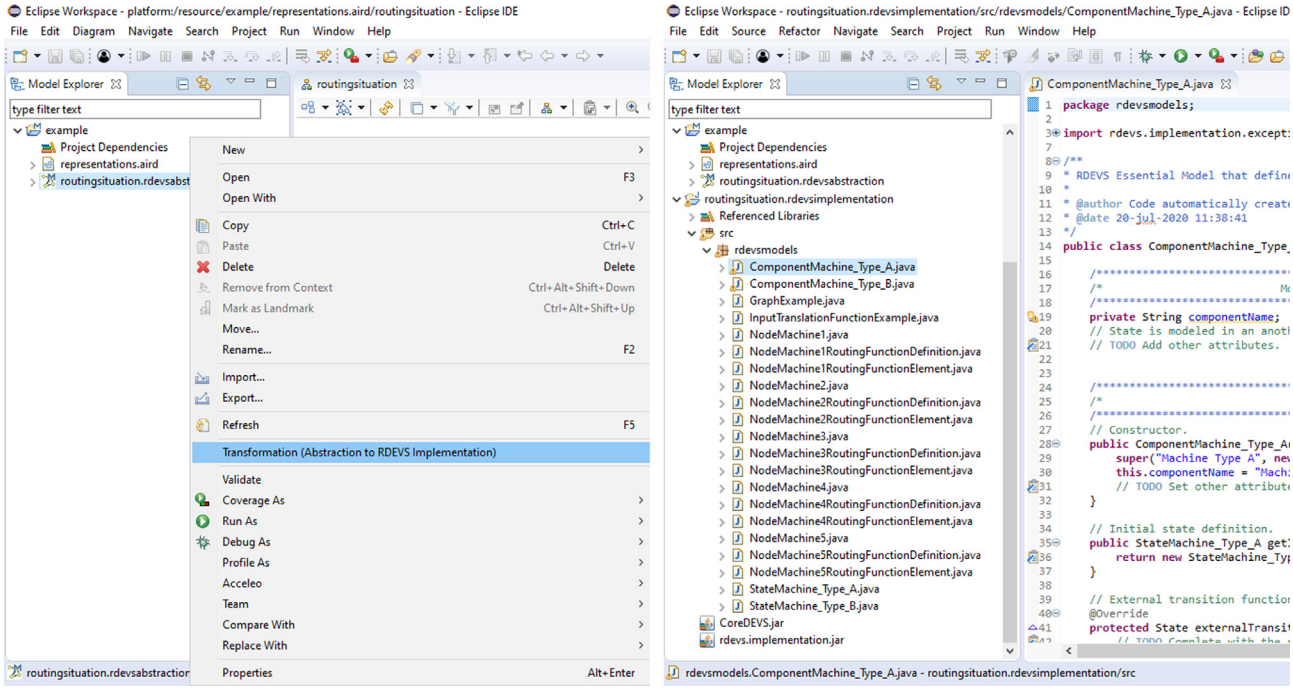


**Fig. 9.** **(a)** At the left, the M&S transformation process executed over the routing situation; **(b)** At the right, a new Java project is created as result of performing the transformation process.

So, the modeler obtains a set of discrete-event simulation models for the original routing situation (in which only some pieces of code are missing) without requiring any other information than the routing situation description. Then, it is clear that the M&S tool provides a fast modeling and implementation solution.

Once the domain-components are implemented using the domain description; the simulation models can be executed. Fig. 10 shows how these implementation models are executable using DEVSJAVA Viewer. The Viewer is configured to execute the model "GraphExample" included in the package "rdevsmodels". Given that such a model is a RDEVS network model, each one of the gray boxes depicts a routing model. Each routing model matches some Node of the Graph. For example, "Machine 1" (Fig. 8) is implemented in the Java class named "NodeMachine1.j

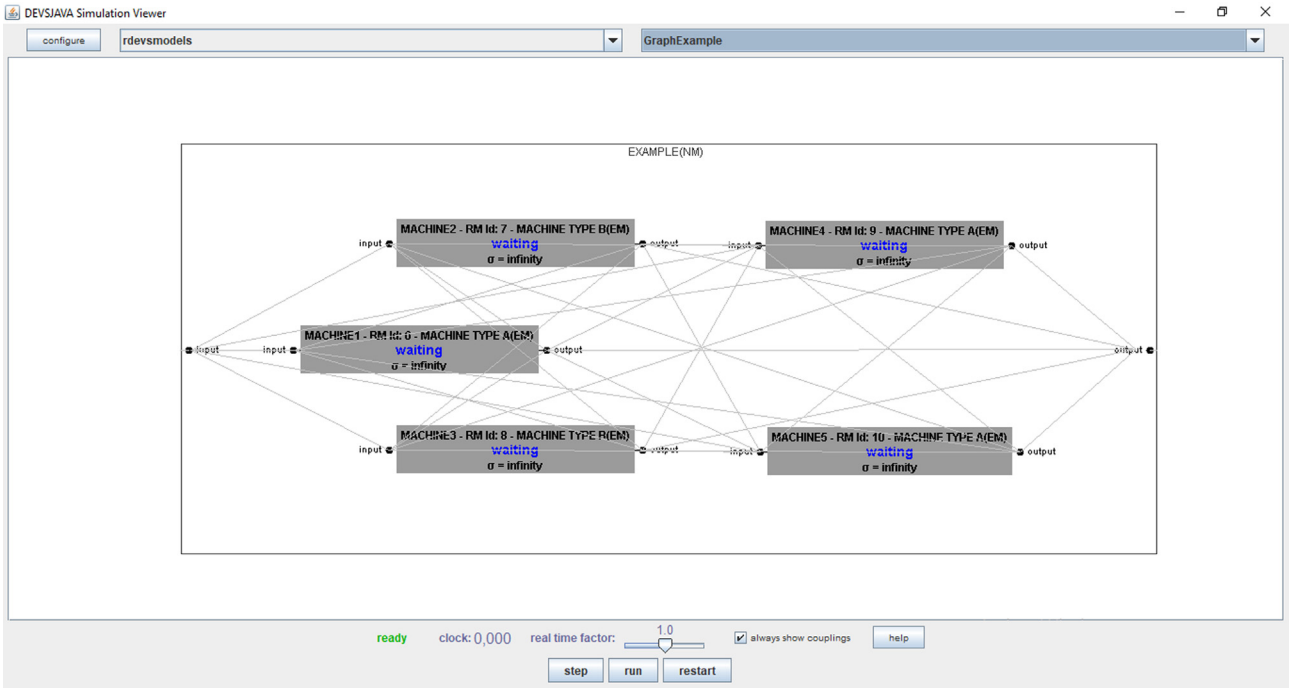


**Fig. 10.** Screenshot of DEVSJAVA Viewer for the example implemented with the M&S software tool. The configuration is detailed at the top of the screen. At the bottom, the execution controls are available.

ava". Such class is depicted in the box labeled as "MACHINE 1 – RM Id: 0 – MACHINE TYPE A(EM)". This label indicates that the routing model embeds "Machine Type A" using zero as the routing identifier.

As Fig. 10 shows, the representation of this model is very similar to the abstraction model representation (Fig. 8) but, also, to the original routing situation (Fig. 7). These equivalences allow understanding all the M&S stages in the same way. The M&S software tool has been successfully used for the study of routing situations related to SE and electric power systems fields.

The models obtained following the MDE modeling levels provide a full solution for the modeling and implementation of routing simulation models. The MDE modeling levels are mapped into implicit models commonly used in the M&S field. By nature, simulation is a technical field [6]. Then, commonly, these models are "implicit" because conceptual modeling tasks are mismatch in M&S processes.

The Eclipse plug-in developed to integrate these models allow following the abstraction-formalization-implementation approach as a new type of modeling task-based in distinct (but related) conceptual modeling perspectives over routing situations. Simulation practitioners do not have to worry about building such conceptualizations. Employing the MDE approach, the M&S tool guides this conceptualization process from abstraction models to well-defined implementations.

## 6. Conclusions and future work

In this paper, we present a novel application of the MDE approach with aims to guide the development of discrete-event simulation models for routing situations. Following this approach, metamodels are defined as abstraction, formalization, and implementation of the desired simulation models. A M&S graphical software tool that employs these metamodels is developed as a plug-in for Eclipse IDE. The final software tool allows building automatic solutions based on discrete-event simulation models for routing situations through the graphical definition of a graph model.

The abstraction-formalization-implementation approach used over 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 clear separation of the routing situation structure and the M&S logic. While the routing situation structure is directly mapped to the RDEVS models (that ensures correct implementation of the routing functionality), the M&S logic is passed to the modelers (with aims to only define the behavior of domain-components). This separation of concerns improves the modifiability of simulation models because it leads to modular designs with low coupling and high cohesion. *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 situations (if routing policies are modified). On the other hand, the utility provided by the extension points of the RDEVS software framework allows maintaining the routing functionality implementation as an isolated module. Changes performed over this functionality will improve its execution but will not require changes in the extensions (that is, the models designed from the transformation process).

The results obtained from the transformation process allows getting RDEVS solutions from graph models. Given that the construction of simulation models for complex systems frequently require to solve situations where components interact following a selective mechanism, our graph model is useful when such a mechanism is defined as part of the situation description. Our modeling strategy allows building scenario descriptions based on graph models with aims to separate the domain component behaviors from the selective mechanism description (i.e., the routing

process description). Then, the modeler can have a simulation model without the need to codify any routing implementation. The main benefits are the reduction of implementation times and the insurance of the simulation model correctness with respect to RDEVS formalism.

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 simulation models for routing situations and, therefore, the modeling tasks could be performed by anyone that knows the problem domain. Of course, M&S experts will be needed for defining the implementation of domain-components to execute the final simulation. To deal with such a definition, new connections to the levels of MDE will be worked in future researches. Still, with the M&S software tool, the M&S process of routing problems is enhanced.

Even when this paper is centered on routing situations, the use of the MDE allows developing new types of M&S products centered in this modeling approach.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work was supported by UTN [Grant Numbers EIUTIFE0003803TC, SIUTIFE0005273TC]; and CONICET [Grant Numbers PIP 112-20170101131CO; PUE 22920160100132CO].

## References

[1] S. Robinson, R. Brooks, K. Kotiadis, D. Van Der Zee, Conceptual Modeling for Discrete-Event Simulation. CRC Press, 2010.
[2] G. Guizzardi, G. Wagner. Conceptual simulation modeling with Onto-UML. In: Proceedings of the Winter Simulation Conference, 5, 2012.
[3] P.T. Eugster, P.A. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Comput. Surv. 35 (2) (2003) 114–131, https://doi.org/10.1145/857076.857078.
[4] B. Zeigler, A. Muzy, E. Kofman, Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations, Academic Press, 2018.
[5] G. Wainer, P. Mosterman, Discrete-Event Modeling and Simulation: Theory and Applications, CRC Press, Taylor and Francis, 2010.
[6] S. Robinson, Conceptual modelling for simulation: Progress and grand challenges, J. Simulation 14 (1) (2020) 1–20, https://doi.org/10.1080/17477778.2019.1604466.
[7] S. Robinson, Conceptual modelling for simulation Part I: definition and requirements, J. Operational Res. Soc. 59 (3) (2008) 278–290, https://doi.org/10.1057/palgrave.jors.2602368.
[8] S. Robinson, Conceptual modeling for simulation: issues and research requirements. In: Proceedings of the 2006 Winter Simulation Conference, 2006, pp. 792–800.
[9] G. Guizzardi, G. Wagner, Towards an ontological foundation of discrete event simulation, in: Proceedings of the 2010 Winter Simulation Conference, 2010, pp. 652–664.
[10] D. Cetinkaya, A. Verbraeck, M. Seck, MDD4MS: a model driven development framework for modeling and simulation, in: Proceedings of the 2011 Summer Computer Simulation Conference, 2011, pp. 113–121.
[11] D. Cetinkaya, A. Verbraeck, M. Seck. A metamodel and a DEVS implementation for component based hierarchical simulation modeling. In: Proceedings of the 2010 Spring Simulation Multiconference, 170, 2010.
[12] G. Zacharewicz, C. Frydman, N. Giambiasi, G-DEVS/HLA Environment for Distributed Simulations of Workflows, Simulation 84 (5) (2008) 197–213, https://doi.org/10.1177/0037549708092833.
[13] D. Cetinkaya, A. Verbraeck, M. Seck, Model Transformation from BPMN to DEVS in the MDD4MS Framework, in: Proceedings of the 2012 Symposium on Theory of Modeling & Simulation, 2012, pp. 304–309.
[14] H. Bazoun, Y. Bouanan, G. Zacharewicz, Y. Ducq, H. Boye. Business process simulation: transformation of BPMN 2.0 to DEVS models. In: Proceedings of the Symposium on Theory of Modeling & Simulation, 20, 2014.
[15] M. Blas, S. Gonnet, H. Leone, Routing Structure over Discrete Event System Specification: A DEVS Adaptation to Develop Smart Routing in Simulation Models, in: In: Proceedings of the 2017 Winter Simulation Conference, 2017, pp. 774–785.
[16] B.P. Zeigler, Closure Under Coupling: Concept, Proofs, DEVS Recent Examples, in: In: Proceedings of the Spring Simulation Multiconference, 2018, pp. 1–6.
[17] The Eclipse Foundation. Eclipse, 2020. https://www.eclipse.org/., (accessed 23rd April 2020).
[18] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Synthesis Lect. Software Eng. 1 (1) (2012) 1–182, https://doi.org/10.2200/S00441ED1V01Y201208SWE001.
[19] OMG. MDA Guide, 2003. http://www.omg.orglcgi-bin/doc?omgl03-06-01.
[20] A. Alshareef, H. Sarjoughian, B. Zarrin, B. Activity-based DEVS modeling. Simulation Modelling Practice and Theory 82(1) (2018) 116-131.
[21] S. Mittal, U. Durak, T. Ören. Guide to Simulation-Based Disciplines, Springer, 2018.
[22] H. Folkerts, T. Pawletta, C. Deatcu, B. Zeigler. Automated, reactive pruning of system entity structures for simulation engineering. In Proceedings of the 2020 Spring Simulation Conference, 61, 1-12, 2020.
[23] T. Ören, S. Mittal, U. Durak. A Shift from Model-Based to Simulation-Based Paradigm: Timeliness and Usefulness for Many Disciplines. Int. J. Computer Software Eng., 3 (1) (2018) 126.
[24] G. Kapos, V. Dalakas, A. Tsadimas, M. Nikolaidou, D. Anagnostopoulos, Model-based system engineering using SysML: Deriving executable simulation models with QVT, in: In: Proceedings of the 2014 IEEE International Systems Conference, 2014, pp. 531–538.
[25] V. Neto, W. Manzano, M. Kassab, E. Nakagawa, Model-based engineering & simulation of software-intensive systems-of-systems: experience report and lessons learned, in: In: Proceedings of the 2018 European Conference on Software Architecture, 2018, pp. 1–7.
[26] Umar Farooq, Gabriel Wainer, Bengu Balya, DEVS modeling of mobile wireless ad hoc networks, Simul. Model. Pract. Theory 15 (3) (2007) 285–314, https://doi.org/10.1016/j.simpat.2006.11.011.
[27] A. Alshareef, H. Sarjoughian. DEVS specification for modeling and simulation of the UML activities. In: Proceedings of the Symposium on Model-driven Approaches for Simulation Engineering, article 9, 2017.
[28] A. Alshareef, H. Sarjoughian. Metamodeling activities for hierarchical component-based models. In: Proceedings of the Theory of Modeling and Simulation Symposium, 2, 2019.
[29] I. Dávid, H. Vangheluwe, Y. Van Tendeloo, Translating engineering workflow models to DEVS for performance evaluation, in: In: Proceedings of the 2018 Winter Simulation Conference, 2018, pp. 616–627.
[30] T. Antoine-Santoni, J.F. Santucci, E. De Gentili, B. Costa, Discrete Event Modeling and Simulation of Wireless Sensor Network Performance, Simulation 84 (2-3) (2008) 103–121, https://doi.org/10.1177/0037549708091641.
[31] M. Blas, S. Gonnet, H. Leone, Building simulation models to evaluate web application architectures, In: Proceedings of the 2016 XLII Latin American Computing Conference (CLEI), 2016.
[32] M. Blas, S. Gonnet, H. Leone, B. Zeigler, A conceptual framework to classify the extensions of DEVS formalism as variants and subclasses, in: In: Proceedings of the 2018 Winter Simulation Conference, 2018, pp. 560–571.
[33] O. Dalle, J. Ribault, J. Himmelspach, Design Considerations for M&S Software, in: In: Proceedings of 2009 Winter Simulation Conference, 2009, pp. 944–955.
[34] M. Hitz, B. Montazeri, Measuring Coupling and Cohesion in Object-Oriented Systems, in: In: Proceedings of the International Symposium on Applied Corporate Computing, 1995, pp. 25–27.
[35] Yentl Van Tendeloo, Hans Vangheluwe, An evaluation of DEVS simulation tools, Simulation 93 (2) (2017) 103–121, https://doi.org/10.1177/0037549716678330.
[36] Federico Bergero, Ernesto Kofman, PowerDEVS: a tool for hybrid system modeling and real-time simulation, Simulation 87 (1-2) (2011) 113–132, https://doi.org/10.1177/0037549710368029.
[37] Matías Bonaventura, Gabriel A Wainer, Rodrigo Castro, Graphical modeling and simulation of discrete-event systems with CD++Builder, Simulation 89 (1) (2013) 4–27, https://doi.org/10.1177/0037549711436267.
[38] Gabriel Wainer, CD++: a toolkit to develop DEVS models, Softw: Pract. Exper. 32 (13) (2002) 1261–1306, https://doi.org/10.1002/spe.482.
[39] L. Capocchi, J.F. Santucci, B. Poggi, C. Nicolai, DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems, in: In: Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2011, pp. 170–175.
[40] G. Quesnel, R. Duboz, E. Ramat, M. K. Traore. VLE: A Multimodeling and Simulation Environment. In: Proceedings of the Summer Simulation Multiconference, 2007, 367-374.
[41] J. De Lara, H. Vangheluwe, AToM 3: A Tool for Multi-formalism and Meta-modelling, in: In: International Conference on Fundamental Approaches to Software Engineering, 2002, pp. 174–188.
[42] A. Levytskyy, E. Kerckhoffs, E. Posse, H. Vangheluwe, Creating DEVS components with the meta-modelling tool AToM3, in: In: 15th European Simulation Symposium (ESS), 2003, pp. 97–103.
[43] A. D'Ambrogio, D. Gianni, J. Risco-Martín, A. Pieroni. A MDA-based approach for the development of DEVS/SOA simulations. In: Proceedings of the 2010 Spring Simulation Multiconference, 142, 2010.
[44] R.E. Johnson, B. Foote, Designing Reusable Classes, J. Object-Oriented Programming 1 (2) (1988) 22–35.

[45] K. Madsen, Five Years of Framework Building: Lessons Learned. In: Companion of the Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, 345–352, 2003.
[46] ACIMS (Arizona Center for Integrative Modeling and Simulation). 2005. DEVSJAVA. http://acims.asu.edu/software/devsjava/, (accessed 23rd April 2020).
[47] B. Klatt, K. Krogmann, Software Extension Mechanisms, in: In: Proceedings of the International Workshop on Component-Oriented Programming, 2008, pp. 11–18.
[48] T. Ören, The Importance of a Comprehensive and Integrative View of M&S, in: In: Proceedings of the Summer Simulation Multiconference, 2007, pp. 996–1006.
[49] M. Nikolaidou, V. Dalakas, L. Mitsi, G.D. Kapos, D. Anagnostopoulos, A SYSML Profile for Classical DEVS Simulators, in: In: Proceedings of the International Conference on Software Engineering Advances, 2008, pp. 445–450.
[50] L. Touraille, M.K. Traoré, D.R. Hill, A Model-Driven Software Environment for Modeling, Simulation and Analysis of Complex Systems, in: In: Proceedings of the Spring Simulation Multiconference, 2011, pp. 229–237.
[51] G. Wainer, Discrete-Event Modeling and Simulation: A Practitioner's Approach, CRC Press, 2017.
[52] The Eclipse Foundation: Eclipse Modeling Project. Eclipse Modeling Framework, 2020. https://www.eclipse.org/modeling/emf/, (accessed 23rd April 2020).
[53] The Eclipse Foundation. Sirius, 2020. https://www.eclipse.org/sirius/, (accessed 23rd April 2020).
[54] The Eclipse Foundation. Acceleo, 2019. https://www.eclipse.org/acceleo/, (accessed 23rd April 2020).