



DEVS-based formalism for the modeling of routing processes

María Julia Blas¹ · Horacio Leone¹ · Silvio Gonnet¹

Received: 22 May 2020 / Revised: 8 September 2021 / Accepted: 16 September 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract

The Discrete Event System Specification (DEVS) is a modular and hierarchical Modeling and Simulation (M&S) formalism based on systems theory that provides a general methodology for the construction of reusable models. Well-defined M&S structures have a positive impact when building simulation models because they can be applied systematically. However, even when DEVS can be used to model routing situations, the structures that emerge from this kind of problem are significant due to the handling of the flow of events. Often, the modeler ends with a lot of simulation models that refer to variants of the same component. The goal of this paper is to analyze the routing process domain from a conceptual modeling perspective through the use of a new DEVS extension called Routed DEVS (RDEVS). The RDEVS formalism is conceptually defined as a subclass of DEVS that manages a set of identified events inside a model network where each node combines a behavioral description with a routing policy. In particular, we study the modeling effort required to solve the M&S of routing problems scenarios employing a comparison between RDEVS modeling solutions and DEVS modeling strategies. Such a comparison is based on measures that promote the capture of the behavioral complexity of the final models. The results obtained highlight the modeling benefits of the RDEVS formalism as a constructor of routing processes. The proposed solution reduces the modeling effort involved in DEVS by specifying the event routing process directly in the RDEVS models using design patterns. The novel contribution is an advance in the understanding of how DEVS as a system modeling formalism supports best practices of software engineering in general and conceptual modeling in particular. The reusability and flexibility of the final simulation models, along with designs with low coupling and high cohesion, are the main benefits of the proposal that improve the M&S task applying a conceptual modeling perspective.

Keywords Routed Discrete Event System Specification · Conceptual modeling · Modeling effort · Routing problem · Design patterns

1 Introduction

A formalism provides a set of conventions for specifying a class of objects in a precise, unambiguous, and paradigm-free manner. In particular, the Discrete Event System Specification (DEVS) is a modular and hierarchical Modeling

and Simulation (M&S) formalism based on systems theory that provides a general methodology for the construction of reusable models [1]. Several applications of DEVS can be found in the literature. For example, in the mobile application field, DEVS models have been used to design mobile application behaviors [2] and to compare the performance of distinct network architectures [3]. In supply chain management, the authors of [4] propose a strategy to generate DEVS and Linear Programming models semi-automatically from industry-scale relational databases. A combination of micro- and macro-views of biological systems with DEVS is proposed in [5, 6] with aims to improve the M&S in computational biology. Since it is very costly to construct a real battle field situation and to verify the performance of military devices, in [7] the authors use the DEVS formalism to define a robotic vehicle model to conduct tests through the simulation of several scenarios. In this case, the final

Communicated by Ketil Stølen.

✉ María Julia Blas
mariajuliablas@santafe-conicet.gov.ar
Horacio Leone
hleone@santafe-conicet.gov.ar
Silvio Gonnet
sgonnet@santafe-conicet.gov.ar

¹ Instituto de Diseño y Desarrollo INGAR (Universidad Tecnológica Nacional, Consejo Nacional de Investigaciones Científicas y Técnicas), Avellaneda 3653, Santa Fe, Argentina

model was embedded in a tank shaped robot. Finally, in the software engineering field, the DEVS formalism has been used to support the evaluation of software architectures [8], provide formal specifications for real-time systems [9], and model web user behaviors [10]. Therefore, DEVS is a popular formalism for modeling complex dynamic systems using a discrete-event abstraction [11]. However, although effective conceptual modeling is a vital aspect of a simulation study, it is probably the most difficult and least understood. 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 [12].

During the design phase of any DEVS model, the modeler designs a simulation model that tries to replicate the operation of a real-world or imaginary system over time to achieve a goal. That is, the problem owner expresses its needs, and the modeler makes an abstraction of the problem to prepare a conceptual model for the simulation study. The conceptual models generally describe the structure and the behavior of the system independent from the implementation details [13]. Frequently, as part of such modeling, the modeler defines several views of the problem. For example, in [14] the authors deal with the M&S of systems described according to several levels of abstraction and levels of granularity. In this context, one recurrent problem during the M&S with DEVS is the need to design an implicit routing process over a discrete-event simulation model that solves a predefined primary goal. For example, when simulation models are designed to evaluate the quality of software architectures, the effectiveness of functionalities is the primary goal. Still, given that components are linked by architecture interactions, the routing of user requests among components becomes an issue to be solved during the M&S task [15, 16]. Another case is frequently observed in the manufacturing domain where real-life scenarios are composed of several machines with the same behavior replicated in multiple work stations that send/receive different kinds of jobs to be processed. Even when all machines involve the same processing, inputs and outputs vary from one workstation to another due to the flow of jobs. When simulation models are used to represent this domain, the final simulation model is composed of several variants of the same component to deal with the flow of events.

In this context, we define the *routing process* as “the part of a modeling scenario where the components need to interact among them by distinguishing the event sources and destinations to ensure their diffusion into the right model”. In DEVS simulation models, the routing processes are commonly solved as pre-wired connections detailed as part of the coupling specification. That is, even when couplings are explicitly defined, the routing functionality of a component

is hidden into its behavioral description. For example, in the manufacturing domain, if the machine of a workstation sends its outputs to three different destinations (machines), its DEVS model is defined with three output ports. Each output port is connected to the appropriate input port of the other machines. Now, if another workstation uses the same machine but requires five destination points, the previous model is adapted to use five output ports. Again, routes will be defined using appropriate couplings between models. Hence, from the structural perspective, the functionality of the routing process is implicitly defined in the pre-wired connections defined by the couplings. Other DEVS-based solutions involve the modeling of handlers that (explicitly) provides the routing functionality. That is, for the previous manufacturing scenario, the model of the machine used in both workstations is (in each case) coupled with an appropriate model of a handler that manages the routes (e.g., by adding a tag to the events). One way or another, besides solving the primary goal, the modeler needs to solve a routing situation by defining (implicitly or explicitly) a routing process specification. Then, due to the modular and hierarchical nature of DEVS, such a routing process specification becomes a systematic problem that is modeled following a predefined structure (i.e., an implicit or explicit definition of routes) that can be seen as a *design pattern over the DEVS formalism*.

A *design pattern* is the re-usable form of a solution to a design problem [17]. Design patterns are general, reusable solutions defined from the study of commonly occurring problems within a given context that ensure the success of the modeling task. For example, object-oriented design patterns capture the intent behind a design by identifying objects, their collaborations, and the distribution of responsibilities [18]. In the M&S of routing processes, DEVS-based solutions (either explicitly or implicitly modeled) can be interpreted as design patterns because they capture the structure behind a simulation model design by defining the connections and routes between models. These well-defined design structures have a positive impact when building DEVS simulation models because they can be applied systematically. Hence, these *design patterns* are powerful modeling tools that provide a good start to define new strategies for solving routing problems. When building a simulation model for solving a routing process, the modeler is solving a *routing problem*. Taking advantage of the modular and hierarchical nature of DEVS, a deeper analysis of the DEVS design patterns can help to improve the modeling of routing processes. That is, from the analysis of the explicit and implicit DEVS-based solutions of routing problems, new structures emerge that can be used to define new types of simulation models that improve the M&S of such problems.

This paper presents an adaptation of the DEVS formalism called Routed DEVS (RDEVS), designed to improve the M&S of routing processes over DEVS models. Such an

adaptation is based on the study of the DEVS design patterns from a conceptual modeling view. By following the approach described in [19], the *conceptual modeling problem* serves as a bridge between the routing problem owner and the simulation modeler. In this paper, we propose a *conceptual model* of generic routing processes to analyze the DEVS design patterns exhibit in the literature. Then, we show how the RDEVS formalism acts as a formalization of the elements that compose the conceptual model to improve the modeling task. Our motivation is to reduce the modeling effort when routing processes are defined over DEVS models. To get an estimation of the effort required for building RDEVS models instead of DEVS models, we use a set of metrics that allow classifying the modeling effort in four cases (high, medium, regular, low). These metrics are tested in this paper as a first attempt to measure the behavioral modeling effort. Hence, we align DEVS and RDEVS solutions over the same situation with the aim to compare such modeling efforts. Such an alignment provides a suitable scenario to discuss the comparison between RDEVS models and DEVS design patterns as vehicles for the M&S of routing processes. Hence, the main contributions of the paper are: (i) the routing problem definition through the specification of a conceptual model that abstracts the elements required for its modeling, (ii) the analysis of the DEVS-based solutions for routing processes as design patterns, (iii) the formal specification of the RDEVS formalism, and (iv) the definition and use of metrics to evaluate the modeling effort.

The remainder of this paper is organized as follows. Section 2 introduces the foundations of our work in terms of DEVS extensions, the routing problem description, and the conceptual model designed for studying the composition of routing processes. It also includes the definition of DEVS modeling complexity along with the DEVS design patterns commonly used for modeling routing processes. These design patterns are presented as implicit and explicit routing strategies commonly used when modeling routing processes with DEVS. To overcome the main issues of such design patterns, Sect. 3 presents the formal specification of RDEVS formalism as an extension of DEVS defined using a set of predefined simulation models. Moreover, it details the mapping between the conceptual modeling problem and the RDEVS simulation models to ensure an appropriate modeling construction for routing processes. Hence, this section shows how RDEVS models cover the same concepts that explicit and implicit design patterns described in Sect. 2. The main benefits of using RDEVS models instead of DEVS design patterns are described in Sect. 4. Finally, Sect. 5 is devoted to conclusions and future work.

2 Background and foundations

DEVS is a popular formalism for modeling complex dynamic systems using a discrete-event abstraction [11]. It provides a general methodology for the hierarchical construction of reusable models in a modular way.

In this context, the DEVS formalism defines two kinds of simulation models. An atomic model describes the autonomous behavior of a discrete-event system as a sequence of deterministic transitions between sequential states. Moreover, it also specifies how the system reacts to external input events and how it generates output events. On the other hand, a coupled model describes a system as a structure of coupled components. Such components can be either atomic or coupled models. The connections between components are structured as couplings that denote how components influence each other. Hence, DEVS is a system theoretic-based formalism that provides representation for systems whose input/output behavior can be described by sequences of events. While an atomic DEVS defines the system behavior, a coupled DEVS defines the system structure. Given that DEVS embodies a set of concepts related to systems theory and modeling with aims to describe discrete-event models in terms of their behavior and structure, such notions of systems can be used as support for the M&S of new definitions.

Over the years, several authors have improved the formalism specification through DEVS extensions to solve new types of problems. Such extensions include: Cell-DEVS [20], Dynamic Structure DEVS [21], Fuzzy-DEVS [22], Min–Max-DEVS [23], Multi-level DEVS [24], Parallel DEVS [25], Real-Time DEVS [26] and Vectorial DEVS [27]. Hence, the extensions of DEVS formalism expand the classes of systems models that can be represented in DEVS [1]. In fact, DEVS can serve as a simulation assembly language to which models in other formalisms can be mapped [11]. Moreover, the extensions of DEVS can be interpreted as sub-formalisms derived from the original specification.

However, with a growing number in new extensions of DEVS and an increasing number of problems to be solved using discrete simulation techniques, the DEVS sub-formalisms can be studied as variants and subclasses [28]. From such a perspective, DEVS variants act as a layer below the DEVS formalism that summarizes a set of models useful to specify new types of systems. Although all extensions provide well-defined solutions for specific contexts, the subset of extensions classified as variants tries to solve modeling and simulation issues from the system representation perspective. The variants of DEVS provide feasible solutions for representing new types of systems dynamics that allow building powerful simulation models. For example, the Cell-DEVS [20] formalism combines cellular automata with DEVS theory to describe n-dimensional cell spaces as discrete event models, where each cell is represented as a DEVS atomic

model that can be delayed using explicit timing constructions [29]. On the other hand, DEVS subclasses act as a layer above the DEVS formalism that structures a set of suitable models for solving specific types of problems without requiring dip down into the DEVS structure itself for modeling special functions. The subset categorized as subclasses tries to solve modeling and simulation issues from the problem representation perspective. Subclasses of DEVS provide reliable solutions for structuring efficient models that allow representing structural problems commonly modeled with DEVS. A well-designed model significantly enhances the possibility that a simulation study will be a success [12].

The RDEVS formalism is an adaptation of DEVS conceptually defined as a subclass of DEVS [28]. The core of RDEVS defines a set of DEVS atomic models that use routing information to authenticate senders, receivers, and transactions. Interpreting this incoming information, routing models only accept specific input messages before passing on the message payload to an associated model for processing. Also, routing models are capable of directing processing results to a specific set of receivers. 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 function [1].

The first description of RDEVS was presented in [30].¹ In this paper, we present an adaptation of the original RDEVS specification that provides a general solution for *routing problems*. Even when *routing problems* have been solved in the M&S community for years (as we will show in Sect. 2.3), we have not found an explicit definition of such a problem. Hence, we propose a definition in Sect. 2.1. Here, routing problems are analyzed in terms of the independence of the components. This independence gives a full separation of concerns at modeling time that reduces the modeling effort involved in the DEVS modeling complexity (Sect. 2.2). To prove that RDEVS is better than DEVS for routing problems, we analyze the structure of the *DEVS patterns used in regular routing problems* (Sect. 2.3). The modeling effort of each pattern is studied to get structural criteria (defined as a design metric) for measuring routing solutions.

2.1 The routing problem

Frequently, simulation scenarios require selectively sending/receiving events from one component to another. We have defined the *routing process* as “the part of a modeling scenario where the components need to interact among them by

distinguishing the event sources and destinations to ensure their diffusion into the right model”. Figure 1 presents two *scenarios* that depict different types of routing processes: output routing using internal information (*scenario #1*) and input routing using external information (*scenario #2*).

In *scenario #1*, there are three types of components: *C1*, *C2*, and *C3*. The behavior required is the following: “The component *C1* must alternate its output events between components *C2* and *C3*. That is, if *C1* sends an event to *C2*, the next event will be sent to *C3* and vice versa, after sending an event to *C3*, the next event will be sent to *C2*”. In this situation, component *C1* needs to route its output events to *C2* and *C3* using its internal information. Such internal information gives the next destination for the outgoing event. Then, the routing process is exhibited from the source component (*C1*) into the set of possible destinations (*C2* and *C3*).

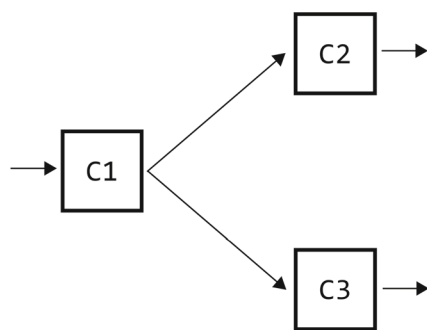
On the other hand, *scenario #2* shows the opposite situation. There are four components in *scenario #2*: *C4*, *C5*, *C6*, and *C7*. Here, the expected behavior is the following: “The component *C7* processes input events from components *C4*, *C5*, and *C6* with different strategies according to their previous processing. That is, component *C7* executes different processes *P1*, *P2*, and *P3* for the events that come from *C4*, *C5*, and *C6*, respectively. Each process has a unique behavior”. Hence, *C7* needs to route its input events from *C4*, *C5*, and *C6* to, respectively, the behavior of *P1*, *P2*, and *P3* using external information. Such external information provides the source component of the incoming event. Then, the routing process is exhibited in the destination component (*C7*) according to the set of possible sources (*C4*, *C5*, and *C6*).

The above *routing process* definition includes both types of routing (*scenario #1* and *scenario #2*) as part of the modeling scenario. Then, the modeling strategy used for solving a full routing process description affects the structure of the final simulation model. When building a simulation model for solving a *routing process*, the modeler is solving a *routing problem*.

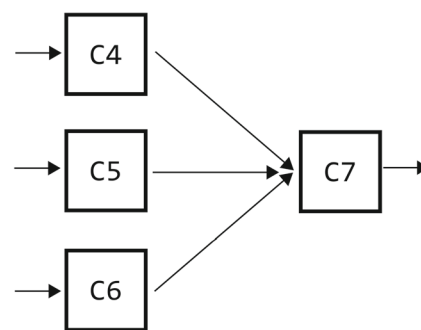
2.1.1 Defining the routing problem as conceptual modeling problem

From a conceptual point of view, the routing problem can be studied as a graph model over a set of predefined components. The aim here is structuring the model as a graph, with the nodes of the graph representing the components, and the edges representing the connections (i.e., the relationships) among these components. Such representations allow modeling a strict separation of concerns between the model’s primary goal and the routing process. For example, the set of predefined components in *scenario #1* is delimited by *C1*, *C2*, and *C3*. The equivalent graph model for such a scenario will

¹ It is important to denote that [30] only includes the RDEVS formalization in set-theoretical notation. In this new version, the RDEVS formalization and the closure under coupling proofs have been improved. Moreover, here we introduce conceptual models to characterize RDEVS definition and compare it with DEVS-based solutions.



(a) Scenario #1: Sending events from one component to several destinations in a selective way.



(b) Scenario #2: Receiving events in the same component from multiple sources in a selective way.

Fig. 1 Scenarios including routing processes in their structure

be composed of three nodes *NC1*, *NC2*, and *NC3* (one node per component) and two edges (one edge per relationship).

Figure 2 presents a UML class diagram [31] that formalizes the routing problem domain in terms of a set of elements commonly identified as part of the routing modeling scenarios. Concepts stereotyped as `<<routing process>>` are used to define the routing process domain. Meanwhile, the concepts stereotyped as `<<scenario goal>>` describe the main elements to be modeled as part of the primary goal of the scenario.

A *Routing Scenario* is structured in, at least, two nodes (*Node* concept). Each *Node* can be conceptually defined as the fundamental unit from which routing processes are made. Then, the relationship *Composed by* defined between *Routing Scenario* and *Node* is detailed as a UML composition. Moreover, nodes are linked by edges (*Edge* concept). Therefore, each *Node* has one input (*Input* concept) and one output (*Output* concept). An *Edge* is an ordered pair of nodes that *Starts at* the *Output* of a *Node* and *Ends at* the *Input* of another *Node*. Then, the concept *Edge* includes the relationships *Starts at* and *Ends at* as mandatory associations with *Output* and *Input* concepts, respectively. However, the *Input* and *Output* of a *Node* can be related to more than one *Edge* (i.e. the associations detailed with multiplicities 1..*).

All the nodes included in a *Routing Scenario* must materialize a component (*Component* concept). A *Component* is an entity of the real world or imaginary system to be modeled that can be characterized by its operation. Such an operation is described as a set of behaviors. Hence, a *Component* definition *Includes* (at least) one behavioral description defined using the *Behavior* concept. The *Materializes* relationship between concepts *Node* and *Component* is mandatory for the *Node* concept (multiplicity 1). However, the same *Component* can be used to build multiple nodes (then, the *Materializes* association for the *Node* concept is detailed with multiplicity 1..*).

When a *Node* *Materializes* a *Component*, the routing information to be used for controlling the input/output of the *Component* related to the *Node* (i.e. the *Routing Policy* concept) needs to be detailed. Such a *Routing Policy* is used to route the input/output events that flow to/from the actual *Node* from/to the other nodes that compose the *Routing Scenario*. For example, in *scenario #1* a *Routing Policy* for *NC1* will involve nodes *NC2* and *NC3*. Even when a new *Node NC1'* can be included in the *Routing Scenario* as a materialization of *Component C1*, the routing policies of *NC1* and *NC1'* may be different. Then, *NC1* and *NC1'* will share the same internal behavior (that is, the *Component C1*) but each node will send/receive events following its own *Routing Policy*. Therefore (as the class diagram of Fig. 2 shows), a *Routing Policy* must be *Defined over* a set of nodes (at least, one *Node*). Moreover, a *Node* included as part of a *Routing Scenario* must be attached to, at least, one *Routing Policy*. Hence, even when the *Routing Policy* is encouraged to manage the flow of events over the edges, its definition must be attached to the *Node* detailed at design time over a *Component*. Then, the *Routing Policy* is defined as an association class linked to the *Materializes* association between *Node* and *Component*.

The conceptual model detailed in Fig. 2 allows an understanding of the modeling relations among the concepts involved in routing situations. At this point, it is important to understand that the routing process definition does not allow universal routing policies at the scenario level. Instead, routing policies must be defined at the node level according to the scenario structure.

2.2 DEVS modeling complexity

Models represent simplifications of a system. Therefore, many models will exist for any particular complex system [32]. For the modeler, the challenge to be solved is the process

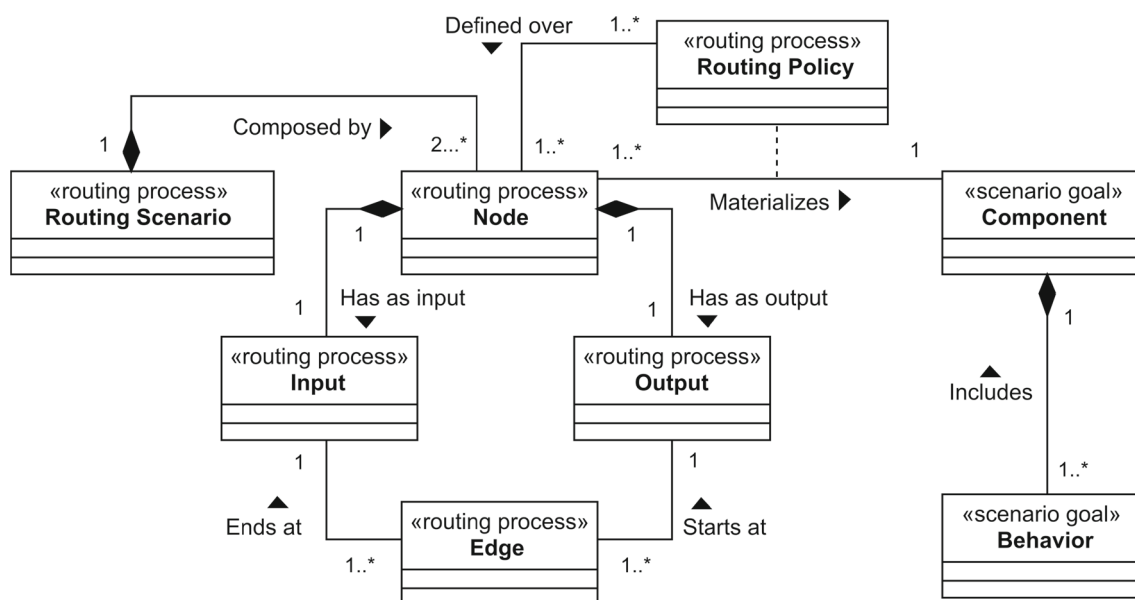


Fig. 2 UML class diagram used for studying the components and their interactions in routing scenario configurations

of exploring a family of hierarchical models and selecting a particular composition from which a fit-for-purpose discrete-event simulation model can be automatically synthesized and executed. For example, in [33] the authors define performance metrics used to guide the modeler to select the most practical model of a real-world system from among a potentially large set.

In this context, assuming that any model can be considered as a set of interconnected components, the overall complexity of a model can be studied as a combination of three elements: the number of components, the pattern of the connections (which components are related), and the nature of the connections (the complexity of the calculations determining the relationships) [34]. Given that the structure of a formalism provides a basis for measuring the complexity of objects in that class [35], the *modeling complexity* of DEVS simulation models can be studied by analyzing DEVS complexity itself.

DEVS models are defined using two abstraction levels: *behavior* and *structure*. Then, *DEVS modeling complexity* should be analyzed using a holistic approach that combines both perspectives as follows:

- The *modeling effort* required for defining the behavior, and
- The *structural complexity* defined over the behavioral descriptions.

Of course, both perspectives are related to modeler skills. The *modeling effort* (or *behavioral modeling effort*) is used as a measure for the complexity of behavioral descriptions. If the modeler does not know the problem domain, the modeling task will be hard. As consequence, behavioral descriptions will be complex. However, if the modeler is a domain expert,

the model design will be easy to understand. Domain experts usually provide an objective view of complex behaviors as a set of simple behaviors. Then, models are structured as a set of behavioral descriptions. Such separation of concerns influences the *structural complexity*. Hence, even when *structural complexity* can be analyzed from an isolated point of view in terms of the model's composition, the *structural complexity* will always depend on the *modeling effort*.

Following the definition presented in Sect. 2.1, our proposal is structuring the routing problem model as a graph. Over this representation, graph theory measures can be used to measure the *structural complexity* of the models. Therefore, at this stage of research, we leave aside the *structural complexity* and only focus our attention on studying the *modeling effort*. Our aim is to study the *modeling effort* attached to distinct simulation models of routing problems to compare DEVS and RDEVs solutions.

2.2.1 Design metrics for studying DEVS modeling effort

Given the need to measure the *modeling effort* in routing problem solutions, two types of measures are taken: *scenario metrics* and *simulation model metrics*. The *scenario metrics* refer to properties related to the routing scenario to be modeled. In this paper, we only use the *number of nodes to be modeled* (defined as N). On the other hand, the *simulation model metrics* refer to the properties of the DEVS models designed by the modeler for solving the scenario. It is important to denote that with aims to model the *behavioral modeling effort*, we employ metrics attached to the *simulation model structure*. As stated in the previous section, DEVS *structure* refers to the definition of a model while DEVS

behavior refers to the pairs of input/output time-indexed trajectories generated by a structure via a simulator. Due to the structure of DEVS, we use three properties: *number of simulation models* (defined as Q), *number of couplings* (defined as K), and *number of ports* (defined as W). Over these metrics, we define two modeling relations (along with feasible indicators) to value the *modeling effort* as a combination of both: Q/N and W/K . From this perspective, *ports* and *couplings* are related to the DEVS structure. So, we employ these measures to evaluate the *modeling effort* considering that the pairs of input/output time-indexed trajectories will be the same in all models compared by the modeling relations Q/N and W/K . These ratios are defined to detect the monolithic level of DEVS configurations and the effectivity of dependencies designed as ports allocated for each coupling. We will test these metrics across our proposal, but deeper analyses will be tested in future research using DEVS models obtained from the literature.

The *number of simulation models* (Q) measures the modeler effort when building a simulation model that imitates the behavior of a component involved in a routing process. As Sect. 2.2 details, domain experts usually provide an objective view of complex behaviors as a set of simple behaviors. Then, a simulation model designed following an appropriate separation of concerns will not be based on monolithic configurations. A monolithic configuration is given when each node is defined as a behavioral simulation model. This means that the *number of nodes* (N) and the *number of simulation models* (Q) are equally valued for the routing problem under analysis. As the *number of simulation models* increases over the same *number of nodes*, the monolithic configuration fades out. Then, the number of simulation models per node (i.e. the Q/N relation) in monolithic configurations is 1. With an appropriate separation of concerns, a good modeler will try to improve the Q/N relation to increasing the average number of models attached to each node. Hence, when studying the closeness to monolithic configurations, the threshold of the Q/N relation can be defined as 1.

Even when the Q/N relation can be used to study the *modeling effort*, it should be analyzed along with the *number of couplings* and the *number of ports* to improve the interpretation of the modeling solution. The *number of couplings* (K) measures the modeler effort when building a simulation model that refers to a component that interacts with other components involved in a routing process. When components interact, the simulation models that represent them are connected using a set of couplings. Intuitively, if the simulation model contains a lot of couplings the modeling effort increases because the modeler needs to consider all possible dependencies among components. In the same direction, if the simulation model contains just a few couplings the modeling task is easier because dependencies among components are restricted into a small set of elements. When

simulation models are designed following an appropriate separation of concerns, the couplings depict either structural or behavioral dependencies among components. Such structural and behavioral dependencies can be interpreted as a UML aggregation or composition [31], respectively, where the model of one class (*parent*) owns the model of another class (*child*). Aggregation implies a relationship where the *child* can exist independently of the *parent* (i.e. structural dependency). Composition implies a relationship where the *child* cannot exist independently of the *parent* (i.e. behavioral dependency).

Either with structural or behavioral dependencies, the *number of couplings* cannot be considered as an isolated metric for measuring *modeling effort*. The *number of ports*² (W) provides a feasible measure that can be combined with the *number of couplings* to estimate the modeling effort. In DEVS models, a coupling is defined as a two-port connection that relates two different models. Then, ideally, the number of ports allocated for each coupling (i.e. the W/K relation) is at most 2. This is because a coupling is defined as a pair of ports. When building a routing problem solution, a good modeler will try to improve the W/K relation to decrease the average number of ports allocated for each coupling. This is, instead of defining a port for each coupling, the modeler will try to reuse ports for several couplings. Hence, the threshold of the W/K relation can be defined as 2.

Tables 1 and 2 introduce both metrics in terms of their definition, minimum and maximum values, and values comparison.

To illustrate the use of these metrics, Figs. 3 and 4 depict two alternative models that solve, respectively, *scenarios #1* (Fig. 1a) and *#2* (Fig. 1b) presented in Sect. 2.1.

Scenario #1 includes three nodes (that is, $N = 3$). In such a scenario, the component $C1$ has two behavioral responsibilities: executing its own operational behavior and distributing its output events between $C2$ and $C3$. *Solution #1* (Fig. 3a) structures the component $C1$ as a single simulation model (monolithic configuration) that performs both behaviors (that is, $Q_{C1} = 1$). Here, the number of couplings is $K = 5$ while the number of ports is $W = 10$. Specifically, the couplings defined in *modelScenario1* depict an *aggregation*³ among *modelC1*, *modelC2*, and *modelC3* for modeling the structural interactions of components $C1$, $C2$, and $C3$. The final value of Q for *solution #1* is $Q_{\text{SOLUTION1}} = 4$. Meanwhile, in *solution #2* (Fig. 3b) the behaviors of component $C1$ are divided into two internal simulation models: *operation*, and *distribution*

² Considering that *i*) all ports have at least one coupling, *ii*) our approach is focused on Classic DEVS, and *iii*) the notion of bag (i.e., a collection of simultaneous external events generated by internal and confluent transitions) on one port is not admitted.

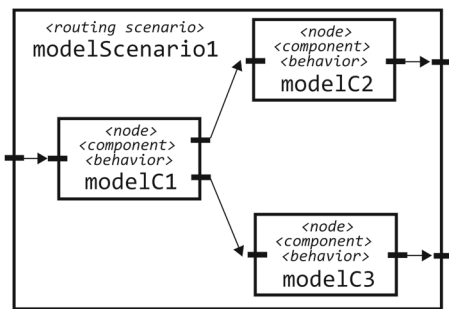
³ Models *modelC1*, *modelC2*, and *modelC3* can be used outside the model *modelScenario1* because each one of them represents a *Component* (Fig. 2).

Table 1 Definition of the metric “ Q/N ”

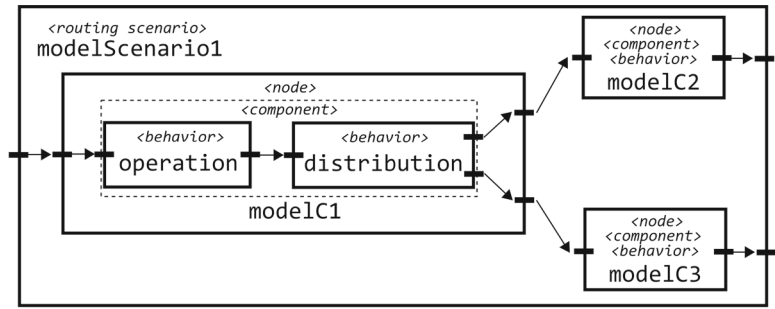
Q/N Property	Description
Definition	Average number of models attached to each node where Q is the number of models and N is the number of nodes
Minimum value	A value of $Q/N = 1$ indicates a monolithic configuration (i.e., each node is represented by a model)
Maximum value	A value of $Q/N > 1$ indicates that the number of models is higher than the number of nodes. Hence, a model composition is used to represent each node
Comparison between values	Given two different measures $m1$ and $m2$, the closest value to 1 will refer to a more monolithic configuration than the other one. A value far from 1 is desirable since it indicates that the modeler has a good modeling approach (i.e., it has performed an accurate separation of concerns)

Table 2 Definition of the metric “ W/K ”

W/K Property	Description
Definition	Average number of ports allocated for each coupling where W is the number of ports and K is the number of couplings
Minimum value	A value of $W/K < 2$ indicates that there are ports that are used to address more than one coupling
Maximum value	A value of $W/K = 2$ indicates that each pair of ports is used to address one coupling (i.e., ports are designed for specific couplings)
Comparison between values	Given two different measures $m1$ and $m2$, the closest value to 2 will refer to a highly structure route solution than the other one. A value far from 2 is desirable since it indicates that the modeler has improved the routes definition (i.e., it has performed an accurate separation of concerns)

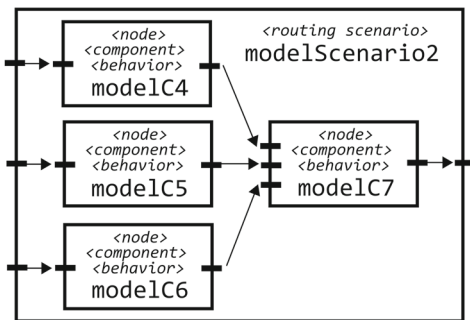


(a) *Solution #1*: Model C1 as a monolithic behavioral model.

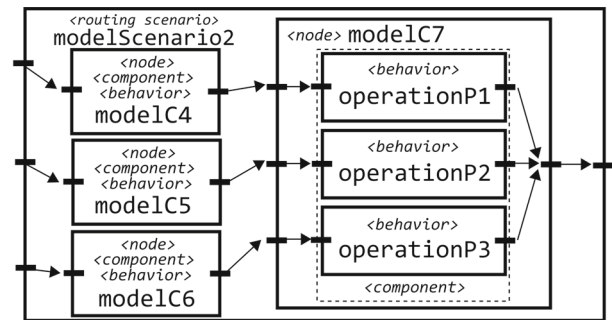


(b) *Solution #2*: Model C1 as a set of behavioral simulation models.

Fig. 3 Possible solutions for building a DEVS simulation model of *scenario #1*



(a) *Solution #1*: Model C7 as a monolithic behavioral model.



(b) *Solution #2*: Model C7 as a set of behavioral simulation models.

Fig. 4 Possible solutions for building a DEVS simulation model of *scenario #2*

(then, $Q_{C1} = 3$). The couplings among such internal models depict a *composition*⁴ among *operation*, *distribution*, and *modelC1* for modeling the behavior required in component *C1*. Moreover, *solution #2* employs 9 couplings ($K = 9$) and 15 ports ($W = 15$). The final value of Q is $Q_{SOLUTION2} = 6$.

From a different point of view of routing processes, the component *C7* of *scenario #2* executes different behavior according to the component that sends the input event (*C4*, *C5*, and *C6*). Such a scenario includes four nodes (that is, $N = 4$). In this case, the simulation model defined as *solution #1* (Fig. 4a) employs a single model (monolithic configuration) to represent all behaviors of component *C7* (that is, $Q_{C7} = 1$). The number of couplings included in this solution is 7 ($K = 7$), while the number of ports is 14 ($W = 14$). The final number of models is $Q_{SOLUTION1} = 5$. Alternatively, *solution #2* (Fig. 4b) splits such behaviors into a set of behavioral models (called *operationP1*, *operationP2*, and *operationP3*) that individually represent each one of the behaviors required in component *C7* (then, $Q_{C7} = 4$). In this solution, the number of couplings is 13 ($K = 13$), and the number of ports is 20 ($W = 20$). The overall number of models included in the solution is $Q_{SOLUTION2} = 8$.

Analyzing the alternative model solutions proposed for both scenarios according to the *modeling effort* metrics, the measures are:

- For *scenario #1*: The modeling effort in *solution #1* (Fig. 3a) is characterized by $Q/N = 1.33$ ($Q = 4$ and $N = 3$) and $W/K = 2$ ($W = 10$ and $K = 5$). Instead, the modeling effort in *solution #2* (Fig. 3b) improves both measures as $Q/N = 2$ ($Q = 6$ and $N = 3$) and $W/K = 1.66$ ($W = 15$ and $K = 9$).
- For *scenario #2*: The modeling effort in *solution #1* (Fig. 4a) is reduced to $Q/N = 1.25$ ($Q = 5$ and $N = 4$) and $W/K = 2$ ($W = 14$ and $K = 7$). Again, *solution #2* (Fig. 4b) improves the measures as $Q/N = 2$ ($Q = 8$ and $N = 4$) and $W/K = 1.53$ ($W = 20$ and $K = 13$).

Table 3 summarizes the criteria adopted for measuring the *modeling effort in routing problems* using the relations Q/N and W/K . Moreover, Fig. 5 depicts the same information as a chart.

According to such valuation, a growing value in the W/K relation (that is, a value closer to 2) increases the modeling effort. If the number of ports allocated for each coupling is closer to 2, the modeler is trying to solve the routing process using pre-wired connections. Independently from the monolithic configuration, the models included in the final

⁴ Models *operation* and *distribution* are not able to be used outside model *modelC1* because each one of them represents a *Behavior* included in a *Component* (Fig. 2).

Table 3 Valuation of the modeling effort to solve routing problems

	Q/N	W/K	Modeling effort
case #HIGH	≈ 1	≈ 2	++
case #MEDIUM	> 1	≈ 2	+
case #REGULAR	≈ 1	< 2	-
case #LOW	> 1	< 2	--

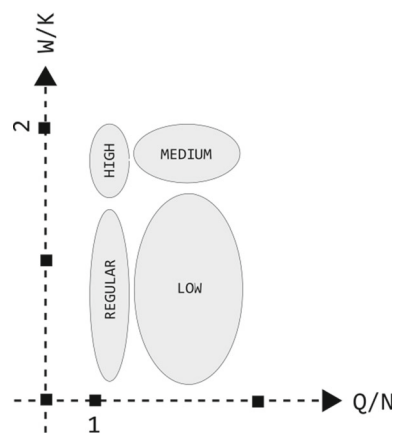


Fig. 5 Chart representation of Table 3

simulation will be handling routing connections as part of their behaviors.

Of course, the modeling effort will be higher if monolithic configurations are employed because a single simulation model will be in charge of handling all behaviors. Therefore, the modeling effort in *case #HIGH* is higher than in *case #MEDIUM*.

Contrary, if the W/K relation is lower than 2, the modeler is trying to improve its solution by leaving the routing responsibility into new components. This reduces the modeling effort. Now, Q/N relation plays an important role. A good modeler will try to improve its design with a proper separation of concerns that not only provides simpler simulation models but, also, improves their reusability. The benefits of reuse should accrue from the reduced time and cost for model development [36]. Therefore, with a Q/N relation closer to 1, the modeling effort increases due to the monolithic configuration (*case #REGULAR*). As the Q/N increases, the modeling effort decreases due to the nodes are modeled as a set of behaviors (*case #LOW*).

Hence, considering the *modeling effort* as a measure for the complexity of behavioral descriptions, the valuation proposed in Table 3 defines four cases with aims to compare such an effort in terms of Q/N and W/K . Then, both relations can be interpreted as *design metrics*. It is important to remember that in this case, the *modeling effort* does not consider *structural complexity* measures (that is, metrics related to the

hierarchical structure). Therefore, an accurate application of these measures must ensure the following:

- The scenario to be modeled is a routing process situation composed of several components explicitly connected through output/input connections.
- The modelers employ similar levels of abstraction for building the simulation models that depict the routing problem.
- All the elements included in the design (i.e., the models, couplings, and ports) have been used to accomplish the simulation purpose.
- The routing policy attached to each node is defined at design time and cannot be changed during the simulation execution (its definition is given by the description of the routing scenario).

2.3 DEVS design patterns for routing process situations

When DEVS is applied for solving routing scenarios, the overall model has two different goals: (i) a primary goal derived from the real world or imaginary system that composes the scenario, and (ii) a routing process goal that helps during the resolution of the primary goal. During the design phase, the modeler must design a DEVS simulation model that achieves both goals. Therefore, the modeler always builds a design that consumes routing information (either by an implicit or explicit specification).

In an *implicit solution*, the models are designed considering the events flow as pre-wired connections. That is, for each possible interaction among components that defines an alternative flow, the model includes a coupling (among specific ports) that is used when the interaction is active. Such an *implicit solution* is the one used in the model solutions detailed in Figs. 3 and 4. For example, the connections among *modelC1*, *modelC2*, and *modelC3* (Fig. 3) are defined using two output ports in the *modelC1*. Each output port of the *modelC1* is specifically linked to the input port of one possible destination. Such destinations are defined according to the routing behavior detailed in component *C1*. Then, each destination is attached to an output port of the source model by a specific coupling. Such couplings allow sending direct events from *modelC1* to *modelC2* and *modelC3* using pre-wired connections. The same *implicit* strategy is used in Fig. 4. However, in these cases, the *modelC7* includes three input ports to distinguish the events that come from *modelC4*, *modelC5*, and *modelC6*. Again, the couplings detailed over these ports can be seen as pre-wired connections. Examples are presented in [37–41].

The use of *implicit solutions* for modeling routing processes can be studied using the UML class diagram depicted in Fig. 2. In an implicit routing, even when a *Node Materi-*

alizes a Component through a *Routing Policy*, the *Routing Policy* is also applied in the *Edge* definition. A *Node (node #1)* decides the output port to be used for sending an event to another *Node (node #2)* using its *Routing Policy*. However, the *Routing Scenario* needs to include an *Edge* that links properly *node #1* and *node #2*. Therefore, the edges need to be defined complying the routing policies required in all nodes. Figure 6 depicts such consideration by including the *Complies* relationship between the concepts *Edge* and *Routing Policy*.

Even when implicit solutions are widely used for routing problems, such solutions do not follow the *low coupling*⁵ guide commonly applied in modular designs. An implicit solution for routing processes in DEVS implies that each one of the models that represent a *Node* will be as specific as its *Routing Policy*. Then, the modeling effort increases because the modeler needs to study the *Routing Policy* to build its design. Moreover, the simulation model that defines the *Routing Scenario* will be also as specific as the routing policies used in the set of nodes that compose it. Hence, a complex *Routing Policy* leads to highly linked specific models that are not completely independent of each other.

Contrary to an implicit *strategy*, *explicit routing solutions* are also encountered in the literature. Such solutions commonly involve a set of handlers that manages the routing functionality. For example, Fig. 7 presents an *explicit strategy* as an alternative solution for the model proposed in Fig. 4b (that is, *modelScenario2*). According to this solution, the model *modelC7* receives all the events produced by models *modelC4*, *modelC5*, and *modelC6* using the same input port. Such an input port is designed as a unique entrance to the model that implements component *C7* (that is, *modelC7*). Then, the model *modelC7* uses a new internal model (called *inputHR*) to distribute the events among the available behaviors (that is, *operationP1*, *operationP2*, and *operationP3*). Examples that use the *explicit routing strategy* for solving routing problems are detailed in [1, 15, 41, 43].

When studying the use of explicit solutions in terms of the UML class diagram depicted in Fig. 2, a new concept *Handler* must be added in the domain to reflect the modeling strategy (Fig. 8). Now, a *Node* uses a *Handler* with aims to apply its *Routing Policy* according to the *Component* description. Such *Handler Manages* the *Input* and *Output* of the related *Node* to restrict the incoming/outgoing events to the *Component* in which the *Routing Policy* is applied. Therefore, the explicit solutions isolate the *Routing Policy* in a specific part of the *Node* (i.e. the *Handler*) giving a complete separation of concerns between the routing responsibility and the *Behaviors* included in the *Component*.

⁵ In software engineering, the coupling is the strength of the relationships between modules [42].

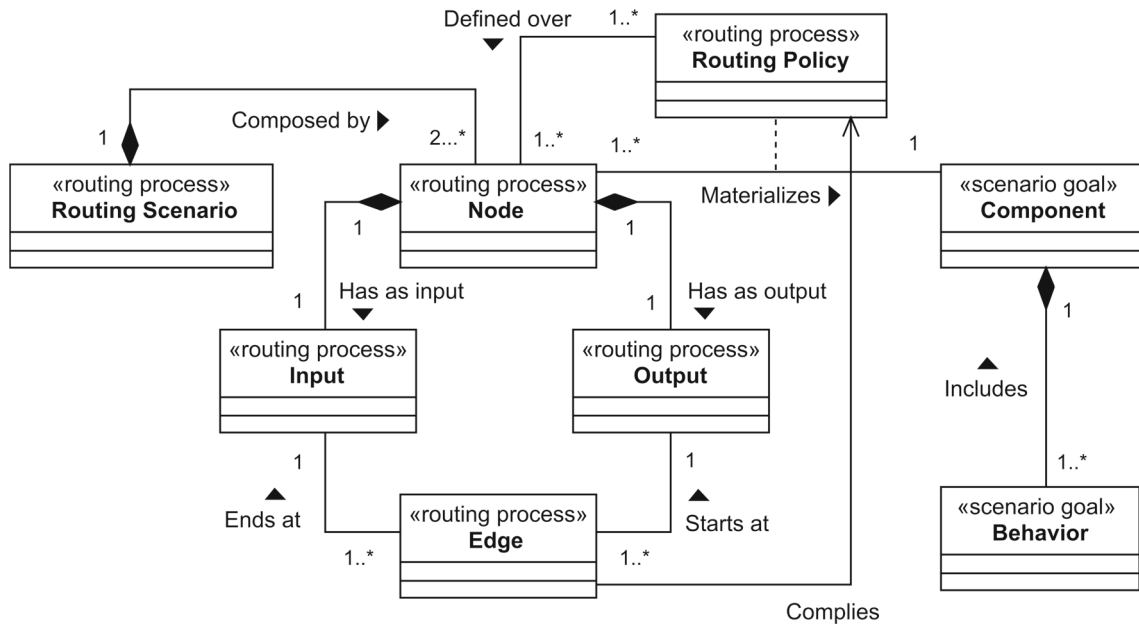
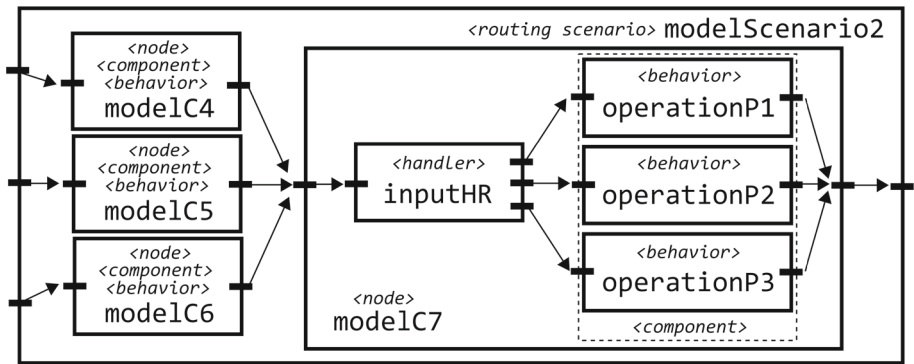


Fig. 6 UML class diagram of the routing problem domain with implicit routing solutions

Fig. 7 Explicit solution for modeling the routing process required in scenario #2



An explicit solution for routing processes in DEVS implies that each one of the handlers included in the nodes will be as specific as the *Routing Policy* attached to its definition. Hence, a complex *Routing Policy* leads to specific handlers but, still, it is independent of the behavioral modeling (related to the *Component*). This particularity gives *low coupling* among models. However, the effort involved in the design of explicit *solutions* is higher than in the equivalent *implicit solution* because the modeler needs to design a larger number of models per node. That is the reason why *implicit solutions* are more common than *explicit solutions*.

Either with an *implicit* or *explicit* solution, the modeler always employs external information to structure the simulation model. Such external information can be interpreted as the routing path to be followed by events. Hence, both solutions can be studied as *design patterns for routing problems*. A *design pattern* is the re-usable form of a solution to a design problem [17]. Therefore, the *implicit* and *explicit* solutions detailed are design patterns over the DEVS formal-

ism that help to build simulation models for routing processes by following a general strategy based on a pattern-oriented modeling approach for designing and developing models of complex systems [44]. Hence, the *design metrics* defined in Sect. 2.2 can be used to analyze their *modeling effort*.

3 The routed DEVS formalism

The RDEVS formalism is an extension of DEVS conceptually defined as a subclass of DEVS that provides a design structure that reduces the *modeling effort* of routing processes over discrete event models [28]. The design patterns detailed in Sect. 2.3 as implicit and explicit solutions leave the routing modeling decision to the modeler. That is, the modeler that builds the design specifies the routing policy following some modeling strategy. When implicit solutions are used, the routing information is hidden in the couplings. When explicit solutions are used, the routing information is hidden

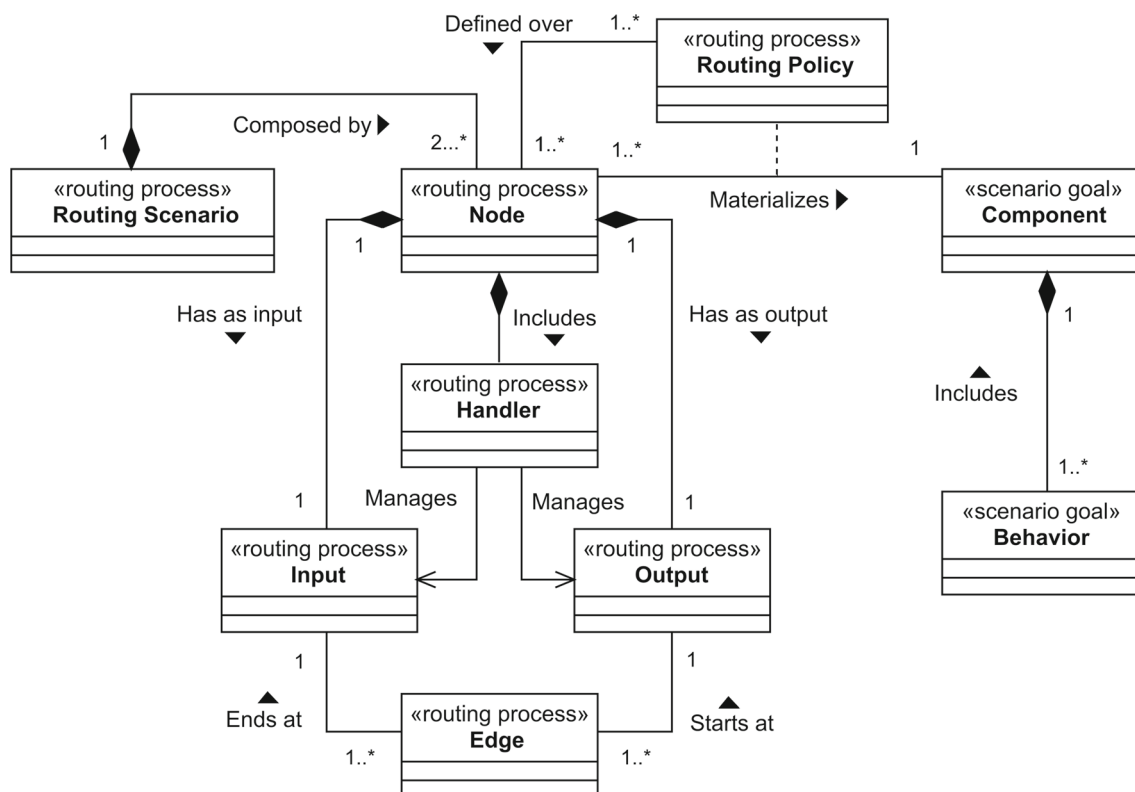


Fig. 8 UML class diagram of the routing problem domain with explicit routing solutions

in the handler definition. In both cases, for modelers who have not been involved in the design, a full understanding of the final simulation model will be tricky without adding any external data related to the couplings foundations.

Instead, the RDEVs models allow defining the routing information explicitly without requiring any behavioral description attached to it. Then, the information related to the routing policy is detailed by the modeler as part of the node definition to authenticate senders and receivers before executing the behavior of the component. The execution of the routing process is built-in RDEVs models. Then, routing models provide an appropriate separation of concerns in terms of the routing process description (i.e., the structure and routing paths) and the behavior of the components. Hence, modelers who have not been involved in the design will only have to understand the behavior of the components.

3.1 The RDEVs models

The RDEVs formalism defines three types of models: *essential*, *routing*, and *network*. Each RDEVs model depicts an abstraction level of some component defined in Fig. 2 as follows:

- A *network model* structures the routing process definition (i.e. the *Routing Scenario* concept). Each *network model*

is composed of a set of *routing models* that represents the entities used to describe routing interactions among components.

- A *routing model* represents an entity involved in the routing process (i.e., the *Node* concept). Each *routing model* is defined as a pair $\{\textit{routing information}, \textit{essential model}\}$ that take care of verifying the *routing policy* over the incoming message and, then, pass on the operative content to the *essential model* for processing.
- An *essential model* refers to a module used to describe a routing entity (i.e. the *Component* concept). Each *essential model* is designed to exhibit the behavior of a domain-specific component.

Hence, the core of the RDEVs approach is using the main components of the routing problem domain to structure the simulation model definition (Fig. 9). RDEVs models embed the definition of the *Components* into the *Nodes* specification that composes the description of the *Routing Scenario*. By embedding the components into the nodes, the formalism improves the modeling task in two ways:

- i) *The design of the behavioral description is only required for the domain-specific components*: The modeler builds simulation models for representing well-known ele-

ments. There is no need to be worried about modeling or implementing routing solutions.

- ii) *The routing policy is isolated from the behavioral description allowing the reuse of simulation models in several scenarios:* The modeler can use the designed set of simulation models without changing their specification with aims to depict different routing processes over the same set of components (only needs to change the routing policy configuration).

The following subsections introduce the formal definition of the RDEVs models using set-theoretic notation.

3.1.1 RDEVs essential model (Component)

The *RDEVs Essential Model* specifies a discrete-event simulation model that exhibits the behavior of a *Component* to be used as part of a *Routing Scenario* (that is, a *routing process component*). The same *Essential Model* can be used to define multiple nodes with different *Routing Policies* over a defined *Routing Scenario*.

An *Essential Model* is equivalent to a DEVS atomic model [1]. Formally, it is defined by the structure

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \tau \rangle$$

where.

- $X \equiv$ set of input events,
- $S \equiv$ set of sequential states,
- $Y \equiv$ set of output events,
- $\delta_{\text{int}}: S \rightarrow S \equiv$ internal transition function,
- $\delta_{\text{ext}}: Q \times X \rightarrow S \equiv$ external transition function, where $Q = \{(s, e) \mid s \in S, 0 \leq e \leq \tau(s)\} \equiv$ total state set,
- $e \equiv$ time elapsed since last transition,
- $\lambda: S \rightarrow Y \cup \emptyset \equiv$ output function,
- $\tau: S \rightarrow R_{0, \infty}^+ \equiv$ time advance function.

The *Essential Model* definitions specify the set of available components that will be used for the M&S of the routing process. To define such *essential models*, the modeler only needs to know the component description (obtained from the domain specification). It does not need to take care of the routing context where the component will be used. Therefore, the modeler could be a domain expert.

3.1.2 RDEVs routing model (Node)

The *RDEVs Routing Model (Routing Model)* defines a discrete-event simulation model that takes action inside the routing process (i.e., a *routing process node*). The *Routing Model* definition employs a *routing process component* as an operational description of its behavior. Hence, an *Essential Model* is embedded in the definition of the *Routing Model*. Moreover, the definition also includes a set of elements used

to describe the *Routing Policy*. Such a policy involves an identifier used to find the *Node* inside the *Routing Scenario*. By using this identifier, the model decides whatever to accept or deny input events and how to route its own output events into other models. The first test is performed when executing the external transition function. The routing decision is taken when executing the output function.

As discussed above, several *Routing Models* can be defined by combining the same *Essential Model* with different *Routing Policy*. The inverse is also true. The same *Routing Policy* can be used in several *Routing Models* with different *Essential Model*. Then, the modeler can design alternative *Routing Scenarios* using the same set of *Components* in distinct *Nodes*.

Formally, the *RoutingModel* is defined by the structure

$$R = \langle \omega, E, M \rangle$$

where.

$\omega = (u, W, \delta_r) \equiv$ routing policy, where.

$u \in N_0 \equiv$ model identifier,

$W = \{w_1, w_2, \dots, w_p \mid w_1, w_2, \dots, w_p \in N_0\} \equiv$ set of identifiers that restricts the nodes from which input events will be accepted,

$\delta_r: S_M \rightarrow T_{OUT} \equiv$ routing function used to direct the output events, where S_M is the state of M and $T_{OUT} = \{t_1, t_2, \dots, t_k \mid t_1, t_2, \dots, t_k \in N_0\} \equiv$ set of identifiers that restricts the nodes towards which output events will be sent,

$E = \langle X_E, S_E, Y_E, \delta_{\text{int}, E}, \delta_{\text{ext}, E}, \lambda_E, \tau_E \rangle \equiv$ RDEVs essential model embedded in R ,

$M = \langle X_M, S_M, Y_M, \delta_{\text{int}, M}, \delta_{\text{ext}, M}, \lambda_M, \tau_M \rangle \equiv$ DEVS atomic model that describes the behavior of the node to be executed during the routing process simulation, where

$X_M = \{(x, h, T_{IN}) \mid x \in X_E, h \in N_0, T_{IN} = \{t_1, t_2, \dots, t_k \mid t_1, t_2, \dots, t_k \in N_0\}\} \equiv$ set of input events identified inside the routing process, with

$x \equiv$ input event defined in E ,

$h \equiv$ sender identifier,

$T_{IN} \equiv$ set of identifiers that represents enabled target models,

$S_M = S_E \equiv$ set of sequential states,

$Y_M = \{(y, u, T_{OUT}) \mid y \in Y_E, u \in N_0, T_{OUT} = \{t_1, t_2, \dots, t_k \mid t_1, t_2, \dots, t_k \in N_0\}\} \equiv$ set of output events identified inside the routing process, with

$y \equiv$ output event generated by E ,

$u \equiv$ sender identifier (that is, the model identifier),

$T_{OUT} \equiv$ set of identifiers that represents enabled target models,

$\delta_{\text{int}, M}: S_M \rightarrow S_M = \delta_{\text{int}, E} \equiv$ internal transition function,

$\delta_{\text{ext}, M}: Q_M \times X_M \rightarrow S_M \equiv$ external transition function that only accepts input events that satisfy one of the following statements: (i) *The event has been sent to R from some allowed model (Clause #1):* This clause is activated when u

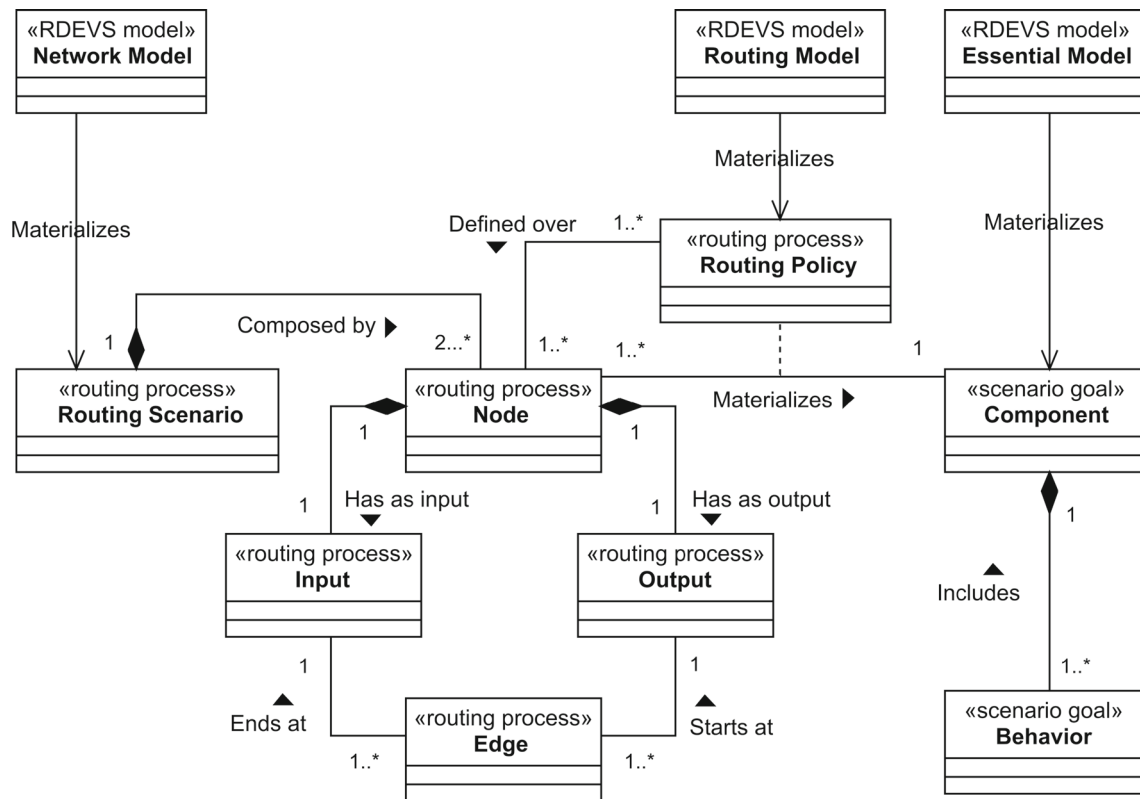


Fig. 9 UML class diagram of the routing problem domain with RDEVS models

belongs to T_{IN} (i.e. the set of identifiers that represents the enabled target models for the incoming event) and the source identifier h (i.e. the sender of the incoming event) is included in W . (ii) *The event comes from an external source (Clause #2)*: This clause is activated when h is zero (that is, the sender identifier of the incoming event is fixed as 0) and u belongs to T_{IN} (i.e. the set of identifiers that represents the enabled target models for the incoming event). (iii) *The model configuration constraints it to accept all input events (Clause #3)*: This clause is activated when the model identifier u is zero and W is configured as an empty set (i.e. there is no restriction related to the achievable senders). Under this clause, the model will exhibit similar behavior to a DEVS atomic model. Then, the external function is defined as

$$\delta_{\text{ext},M}(s, e, x') = \begin{cases} \delta_{\text{ext},E}(s, e_{-c} + e, x) & \text{if } (u \in T_{IN} \wedge h \in W) \vee (h = 0 \wedge u \in T_{IN}) \vee (u = 0 \wedge W = \emptyset) \\ & \text{with } x' = (x, h, T) \text{ and setting } e_{-c} = 0 \text{ after the execution} \\ s & \text{otherwise} \\ \text{Updating the elapsed time accumulated value as } e_{-c} = e_{-c} + e & \end{cases}$$

with.

$Q_M = \{(s, e) \mid s \in S_M, 0 \leq e \leq \tau_M(s)\} \equiv$ total state set,
 $e \equiv$ time elapsed since last transition in M ,
 $e_{-c} \equiv$ time elapsed since last transition in E ,

$\lambda_M: S_M \rightarrow Y_M \cup \emptyset \equiv$ output function that produces events with routing information, defined as

$$\lambda_M(s) = (\lambda_E(s), u, \delta_r(s))$$

$\tau_M: S_M \rightarrow R_{0,\infty}^+ \equiv$ time advance function.

In a *Routing Model*, an event is identified by the sender information (i.e. the model that generates the event as part of its output function). Although natural numbers are used to define the identifiers, such elements can be defined using other types of individualization.

As defined above, the simulation functions of M embed the functions of E to allow the routing process only when

events are accepted inside the *Node*. Such embedding can be understood as a new type of relationship among DEVS models. Besides the structural and behavioral dependencies, the RDEVS formalism includes an embedding dependency in

which the model of one class "uses" the model of another class. Therefore, by using this new type of dependency among models, the formalism details the functions of the *Routing Model* using the *Essential Model* definition. For example, the external transition function is designed as an if-else function to accept only the set of input events that must be processed inside the *Node*. That is, if an input event fulfills some of the defined clauses, the *Routing Model* executes the external transition function of the *Essential Model* to exhibit the behavior of the *Component* embedded in the *Node*. Otherwise, the model "rejects" the incoming event staying in the same state that was before it arrives. As opposed, the internal transitions of the *Routing Model* are governed by the *Essential Model*. Given that the *Node* must exhibit the same behavior as the *Component*, the internal transitions of both models should be the same (that is why the internal transition function of *M* is equivalent to the one defined in *E*). Then, the internal transitions of the *Essential Model* will take place in the *Routing Model*. Moreover, given that before executing an internal transition the model needs to produce an output event, the output function of *M* employs the output function of *E* surrounded with the routing data. Therefore, the events sent by the *Routing Model* include the events produced by *E* along with the model identifier (sender) and the set of enabled destinations obtained from the routing function.

The *Routing Model* definition follows the statement detailed in Sect. 2.2 as "the routing policy attached to each node is defined at design time and cannot be changed during the simulation execution". Hence, the ω definition cannot be dynamic itself. Its values are fixed when the simulation model is designed by following the routing scenario description.

3.1.3 RDEVS network model (Routing scenario)

The *RDEVS Network Model (Network Model)* describes a complex discrete-event simulation model that has a primary goal that includes the resolution of a routing problem (i.e. a *routing process scenario*). To build a *Routing Scenario*, the modeler needs to define a set of *Nodes*. Then, the definition of the *Network Model* includes a set of *Routing Models* and the couplings among them. Such couplings are detailed as all-to-all connections to leave the routing task to the *Nodes*. The *Network Model* specification also involves two special translation functions used to link several models. These functions allow matching events from different *Routing Scenarios*. Then, the design of the *Network Model* is prepared to interact with other *Network Models* or, simply, with DEVS models. The combination of several types of simulation models depends on the simulation goal.

Formally, the *Network Model* is defined by the structure

$$N = \langle X, Y, D, \{R_d\}, \{I_d\}, \{Z_{i,d}\}, T_{in}, T_{out}, Select \rangle$$

where.

$X \equiv$ set of input events,

$Y \equiv$ set of output events,

$D \equiv$ set of identifiers that represent the nodes to be used as part of the routing scenario (i.e. references to *RoutingModels*), where $d \in N_0, \forall d \in D$,

For each $d \in D, R_d$ is a *RoutingModel*, defined as

$$R_d = \langle \omega_d, E_d, M_d \rangle$$

where $u_d = d$,

For each $d \in D \cup \{N\}, I_d$ is the set of influences over d defined as $I_d = \{i / i \in D \wedge i \neq d\} \cup \{N\}$ to maintain the all-to-all couplings,

For each $i \in I_d, Z_{i,d}$ is a translation function between events of i and events of d , where

$$Z_{i,d} = T_{in}, \text{ if } i = N,$$

$$Z_{i,d} = T_{out}, \text{ if } d = N,$$

$$Z_{i,d}: Y_{M,i} \rightarrow X_{M,d} \text{ if } i \neq N \wedge d \neq N,$$

$T_{in}: X \rightarrow \{(x, h, T) \mid x \in X, h \in N_0, T = \{t_1, t_2, \dots, t_k \mid t_1, t_2, \dots, t_k \in N_0\}\} \equiv$ input translation function that takes an external input event and returns an input event identified inside the routing scenario, where

$x \equiv$ input event defined in N ,

$h = 0 \equiv$ sender identifier where zero indicates that the event comes from an external source,

$T \equiv$ set of enabled destinations (identifiers) that must process the input event,

$T_{out}: \{(y, h, T) \mid y \in Y, h \in N_0, T = \{t_1, t_2, \dots, t_k \mid t_1, t_2, \dots, t_k \in N_0\}\} \rightarrow Y \equiv$ output translation function that takes an output event identified inside the routing scenario and returns an external output event, where

$y \equiv$ output event allowed in N ,

$h \equiv$ sender identifier,

$T = \emptyset \equiv$ set of enabled destinations (empty set implies that events go to an external destination),

$Select: 2^D \rightarrow D \equiv$ function for tie-breaking between simultaneous internal transitions.

Following the formal definition, the *Network Model* only comprehends the routing process scenario modeled inside of it. That is why the events that come from external models are not treated as identified events. All the external input events that arrive at the *Network Model* are translated into events with identification inside the routing scenario by using the input translation function. In such cases, given that the sender identifier is not defined as part of the scenario, the input translation function fix h to zero. Any *Routing Model* included in the *Network Model* that receives an event with $h = 0$ should process it according to *Clause #2*. That is, the *Routing Model* verifies that its own identifier is included in T_{IN} prior accept the incoming event. A similar setting strategy is used for output events. Conceptually, the output events of the *Network Model* are defined as identified events with

external destinations. However, given that the routing process only takes place inside the *Network Model*, external identifications are not allowed. Therefore, external destinations are detailed setting T_{OUT} as an empty set (giving “everywhere” as destination). Moreover, all the events that leave the *Network Model* (classified as events with identification inside the routing process) must be transformed into DEVS regular events to keep the *Routing Scenario* isolated from the other simulation models. Such a transformation is performed by executing the output translation function.

3.2 Closure under coupling

Closure under coupling justifies hierarchical construction. A system formalism is closed under coupling if the result of any network of systems specified in the formalism is itself a system in the formalism [1]. However, it also assures that the class under consideration is well-defined and enables checking for the correct functioning of feedback coupled models [45].

To prove that the RDEVS formalism is closed under coupling, two new models are obtained: (i) the *Routing Model* that is behavioral equivalent to the *Network Model*, and (ii) the *Essential Model* that is behavioral equivalent to the *Routing Model*. Appendix A summarizes both demonstrations. The equivalent models obtained in such demonstrations can be used to justify the hierarchical composition because they ensure that the *Network Model* and *Routing Model* can be used to structure hierarchically the *Routing Model* and *Essential Model*, respectively. By transitivity, a dynamic system specified with a *Network Model* can be reduced to a behavioral equivalent *Routing Model* and, then, the *Routing Model* can be transformed into the behavioral equivalent *Essential Model*. Hence, a *Network Model* can be reduced to an equivalent *Essential Model*. Such an *Essential Model* can be used in the new *Routing Model* (following the *Routing Model* definition). Now, a *Routing Scenario* (the *Network Model*) can be used to structure new *Nodes* (the *Routing Models*). Therefore, the hierarchical construction stays within the formalism.

Although equivalent models are not commonly used for closure under coupling demonstrations, such a strategy for RDEVS allows proving that any RDEVS model can be reduced to a DEVS atomic model (because the *Essential Model* is defined as a DEVS atomic model). Then, the RDEVS models can be combined with DEVS models develop larger simulation structures. Moreover, given that RDEVS models are compatible with DEVS models, the RDEVS models can be executed using DEVS simulators.

3.3 About the RDEVS structure

The RDEVS formalism is designed for leveling out the modeling effort of routing problems providing an easier modeling

solution that employs a set of simulation models that are defined in terms of the main elements involved in routing scenarios. In this context, RDEVS models aim to isolate a routing scenario inside the *Network Model*. Such a *Network Model* can be defined to interact with other models. These models can be other routing scenarios (with their own set of nodes and routing policies) or, simply, other DEVS models required as part of the problem. Following this approach, the identifiers used in the *Routing Models* that compose the *Network Model* are unique inside the network definition. However, their definition is not valid for the overall simulation model. That is why the incoming events of the *Network Model* are set with zero as the sender identifier. Zero indicates that the event is coming from outside the network and, then, the *Network Model* must decide which node will be in charge of its processing (employs the input translation function). In the same direction, the outgoing events of the *Network Model* use an empty target list to indicate that the event is going to be sent by all the outputs detailed in the model. Then, these events are sent “everywhere”.

A deeper analysis of RDEVS structure reflects that, even when *Routing Models* are defined using some kind of external information (that is, the identifiers of available senders and possible destinations), the same information was already used when DEVS solutions were applied. When couplings are detailed (in *implicit DEVS solutions*) or handlers are defined (in *explicit DEVS solutions*), the modeler is using external information to get the final design (such as scenario context). In RDEVS solutions, such external information is captured in a single component of the *Routing Model*: the ω definition. This routing policy is defined as a static component (in the same way that couplings and handlers). As in DEVS solutions, such a component should be defined in a way that covers all routing paths required over the scenario. Given that routing paths are defined at design time, there is no way to assess the modeling strategy really fulfills the routing paths required over the scenario.

The static perspective at the routing level is the main difference between RDEVS and Dynamic Structure DEVS (DS-DEVS) [21]. The routing problems studied in this paper employ routing paths defined at design time in terms of the behavior of the components. That is, there is no universal routing policy that acts as a global structure of the simulation model. Moreover, the routing policies definition does not change during the simulation execution and, as consequence, they can be modeled as part of the model’s design. This is the approach used in RDEVS. Instead, in DS-DEVS the routing is encapsulated in the model state. Therefore, it changes at execution time. This behavior is useful when models need to adapt their structure at simulation time by following some specific strategy. However, such adaptability requires defining the simulation models in terms of routing elements (not in terms of the behavior of the components). Even when the

routing problems analyzed with RDEVS could be defined with DS-DEVS, the *modeling effort* will increase without any gain.

Regarding the codification of RDEVS models in general-purpose programming languages, the conceptualizations provided in previous sections can be used as a basis to design object-oriented implementations. Given that DEVS is based on the mathematical theory of systems and works with object-orientation and other computational paradigms [1], the conceptual model depicted in Fig. 9 can be translated to an implementation deployment in, for example, Java or C++ over existing M&S software tools that support the execution of a DEVS abstract simulator. Then, runnable RDEVS models can be implemented following the abstraction of routing processes. Such an implementation is outside the scope of this paper.

4 Benefits of RDEVS formalism over DEVS design patterns for solving routing problems

The success of the modeling task always depends on the modeler's ability to get an appropriate *level of abstraction* in its design. Such a level of abstraction must ensure an adequate number of models, couplings, and ports to improve the model's *reusability*. Moreover, good modeling enables an appropriate *separation of concerns* by giving domain experts the possibility to contribute during the design of the simulation model. This *separation of concerns* improves the model's *modifiability* because it leads to modular designs with *low coupling*⁶ and *high cohesion*⁷ [46–48]. Moreover, it also enhances the model's *maintainability* because the *separation of concerns* reduces the number of (different) models to be developed to get the desired simulation model.

To get appropriate modeling for routing problems, the RDEVS formalism design maintains all these desired properties as part of the model's definition as follows:

- *Level of abstraction and separation of concerns*: Each RDEVS model structures a routing process element defined as part of the core components required for modeling a routing problem. By identifying such elements, the formalism provides an appropriate separation of concerns using the model's goals: *i*) the *Essential Model* is designed

with aims to define the behavior of a component that will be used several times as part of the routing scenario, *ii*) the *Routing Model* is structured with aims to define a node that interacts with other nodes to perform the routing process, and *iii*) the *Network Model* is specified with aims to define the structure of all possible routes (that is, all possible interactions among nodes) as part of the routing scenario definition. These goals allow modelers to easily recognize each RDEVS model as a useful and unique element to be used in the routing problem definition. Moreover, it increases the abstraction level by mapping directly each simulation model into a routing element.

- *Reusability, modifiability, and maintainability*: By following the separation of concerns, the relationships among RDEVS models allow building structural dependencies that reinforce their reusability, modifiability, and maintainability. A specific *Essential Model* definition describes an elemental behavior that can be reused in several nodes (or even as part of DEVS coupled models). Moreover, the same *Network Model* definition can be used to model multiple routing scenarios by only changing the routing policy used in each node. Then, the routing structure is highly modifiable. Finally, given that each simulation model proposed as part of the RDEVS formalism has its own goal, the model maintainability increases as a consequence of the separation of concerns.
- *Low coupling and high cohesion*: Given that connections among models are defined in terms of their structure (not their behavior), the chances that the changes in one model cause problems in other models are really low. Furthermore, the model's understanding is really easy because, per its own definition, each model must be designed to represent a specific routing process element.

Then, the RDEVS formalism provides a feasible solution to modelers to easily build routing processes models. The modeler only uses the RDEVS models as predefined simulation modules that require the specification of the behavior (*Essential Model*), the routing policy (*Routing Model*), and the interactions among components (*Network Model*).

In the following subsections, a generic 3 tier architecture for web applications is used as a case study for modeling a routing problem. The case of the 3-tier architecture has been already studied in [16] to show how discrete-event M&S can be used as support for quality estimation in the early stages of development. In that case, by considering the simulation of software architectures as a routing problem, the RDEVS simulation models were obtained automatically from the software architecture design. Here, we use the same architecture as a routing problem situation to be modeled following two different approaches: DEVS implicit routing solution and the RDEVS formalism. Both solutions are analyzed in terms of the design metrics detailed in Sect. 2.2.

⁶ The *low coupling* is desirable because (i) fewer interconnections among 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 modeler time in understanding the details of other modules [49].

⁷ Cohesion is an important attribute corresponding to the quality of the abstraction captured by the module under consideration. Good abstractions typically exhibit *high cohesion* [46].

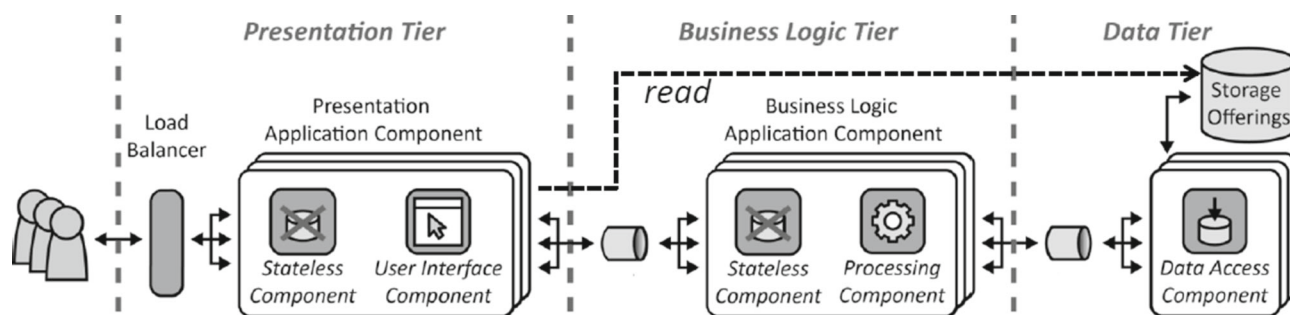


Fig. 10 Generic 3 tier architecture proposed in [51]

4.1 Tier software architectures

The architecture of a system defines that system in terms of computational components and interactions among those components [50]. Most web applications use the 3 tier architecture style (Fig. 10) to arrange its components and the set of connections among them. This style defines three tiers *presentation logic*, *business logic*, and *data handling* composed of predefined functional components named *user interface component* (UIC), *processing component* (PC), and *data access component* (DAC). A functional component is an element that refers to a specific and well-known functional responsibility. As part of the architectural design, the functional components are related by architectural connections. The *stateless component* element defines a synchronization constraint over the functional components and cannot be seen as a functional component itself. Finally, the *storage offering* and *load balancer* represent external components linked to the architecture. Such components are used for specific purposes. The behavior of these components is well-known by the architects and, therefore, does not affect the architecture evaluation.

The 3 tier architectural style allows to scale stateless presentation and compute-intensive processing independently of the data handler (which is harder to scale and often managed by the cloud provider). That is, the style defined in Fig. 10 is a conceptual idea of how the functional components of web architectures should be structured. Architectural designs are obtained when several replicas are defined for each one of the functional components attached to the tiers. Therefore, the 3 tier architectural style defines a routing problem when specific designs are defined. The architectural evaluation of such designs can be performed with discrete event simulation. Figure 11 shows how each part of the architecture design can be mapped into the routing problem domain.

With aims to evaluate the modeling effort in routing problems employing DEVS and RDEVS formalisms, Fig. 12 presents two architectural designs. Each design is considered a routing scenario. Figure 12a shows a scenario composed of 3 replicas of the *user interface component*, 4 replicas of

the *processing component*, and 2 replicas of the *data access component*. Meanwhile, Fig. 12b employs 5 replicas of the *user interface component*, 2 replicas of the *processing component*, and 3 replicas of the *data access component*.

To build discrete event simulation models that provide a feasible solution for both scenarios, each architectural component of the 3 tier architecture should be mapped into an independent simulation model. Such individual models should be able to simulate the behavioral description of the architectural component to fulfill the simulation goal (that is, the architectural evaluation) but also the event redirection actions related to the routing process (that is, the routing of user requests).

4.2 Modeling Effort

4.2.1 Building the simulation models using a DEVS implicit routing solution

Table 4 summarizes the set of simulation models required for modeling the web architectures depicted in Fig. 12 by following the implicit modeling strategy described in Sect. 2.3. Appendix B illustrates the overall DEVS representation for the scenarios detailed in Fig. 12a and b using the elements defined in Table 4.

To keep the proposal analysis, all DEVS models are seen as single simulation models (even when they could be modeled as coupled models). Therefore, the input ports of the model proposed for each component are as specific as the set of nodes from which is receiving its inputs. Moreover, the output ports of the model proposed for each component are as specific as the routing paths where the node is delivering its outputs. Hence, although the behavior of the architectural components used in both scenarios is the same, the simulation models designed for each case should be different to get an appropriate set of routing actions.

When the modeling effort is analyzed in DEVS implicit routing solutions, the *number of simulation models* Q depends explicitly on the *number of nodes to be modeled* N . In this case, an extra simulation model is added to depict the scenario itself. Then, Q is defined as:

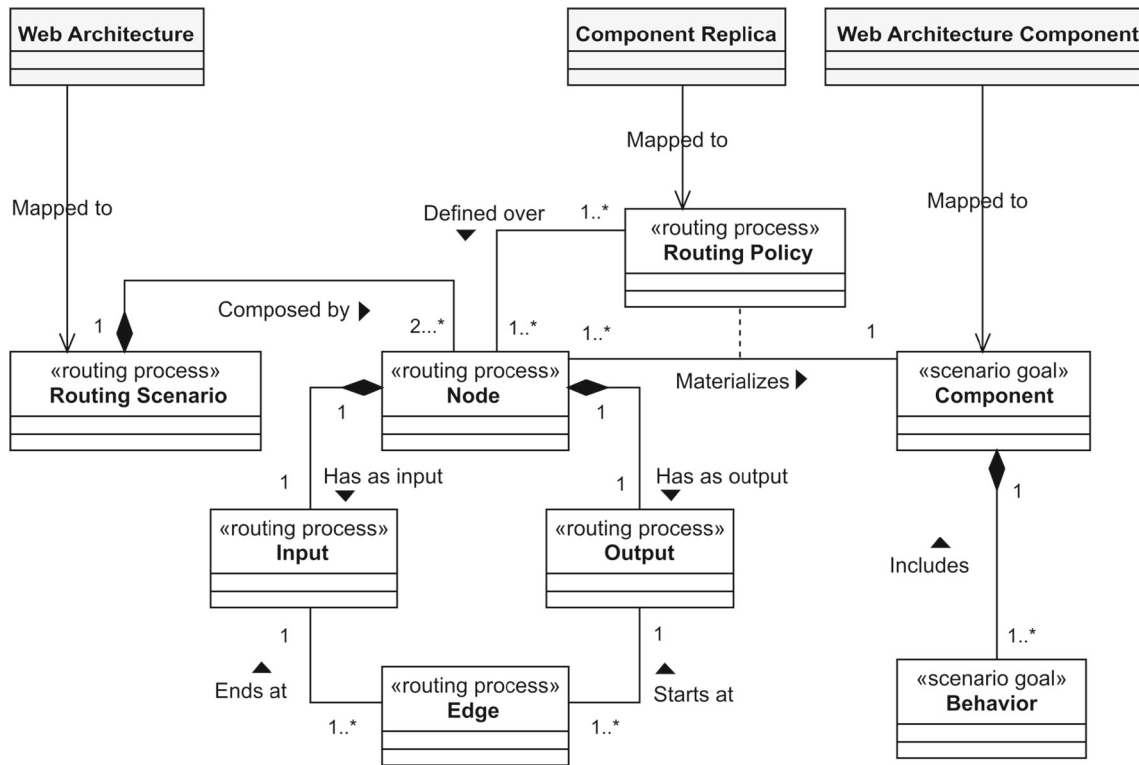


Fig. 11 Mapping web architectures into the routing problem domain

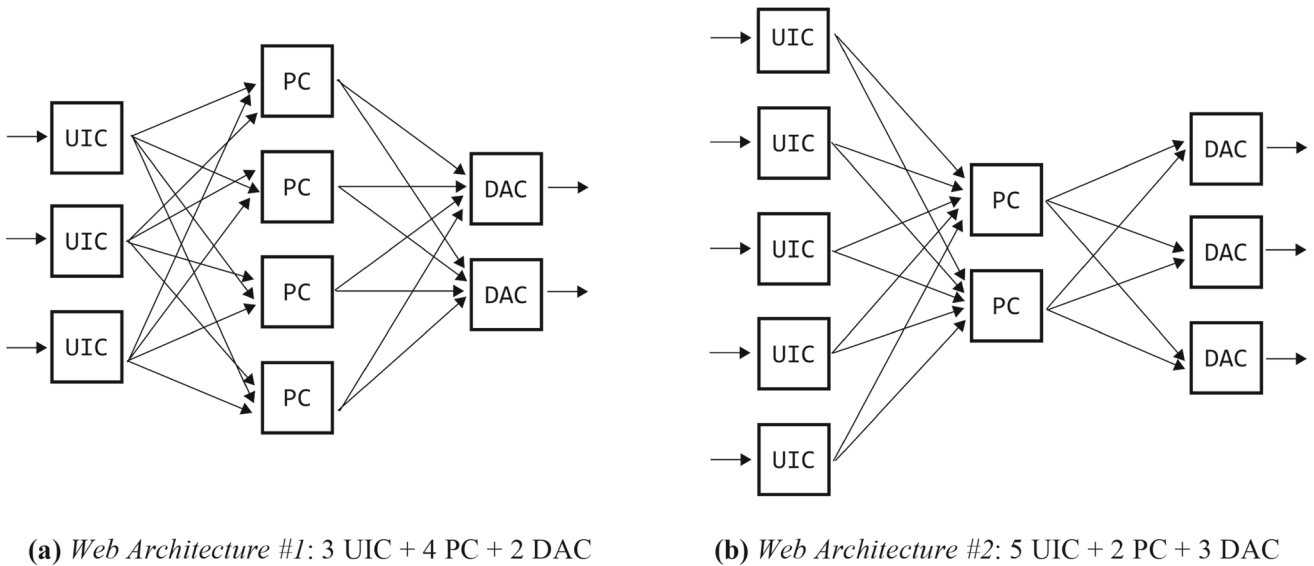


Fig. 12 Web architectures scenarios under evaluation

$$Q_{DEVS} = N + 1$$







Now, the Q/N relation for DEVS implicit solutions is defined as:

$$Q/N_{DEVS} = Q_{DEVS} / N = (N + 1) / N$$

Hence, as the definition exhibits, the final value of the Q/N relation in implicit solutions will always be near 1.

Following the case study, the routing scenarios depicted in Fig. 12 as 3 tiers architectures have $Q/N_{DEVS,WA1} = 1.11$ for *Web Architecture #1* (with $Q_{DEVS,WA1} = 10$ and $N_{WA1} = 9$), and $Q/N_{DEVS,WA2} = 1.1$ for *Web Architecture #2* (with $Q_{DEVS,WA2} = 11$ and $N_{WA2} = 10$).

Table 4 Basic simulation models required for modeling the web architectures scenarios using DEVS

Web architecture	Architectural component	DEVS model representation	Description
Web Architecture #1—Fig. 12a -	UIC		The model is designed as a DEVS model with 1 input port for receiving the user requests and 4 output ports for redirecting the requests (already processed) to a specific instance of the PC component
	PC		The model is designed as a DEVS model with 3 input ports for receiving the requests (processed by some UIC component) and 2 output ports for redirecting the requests (already processed by itself) to a specific instance of the DAC component
	DAC		The model is designed as a DEVS model with 4 input ports for receiving the requests (processed by some PC component) and 1 output port for sending the final resolutions of the initial user requests
Web Architecture #2—Fig. 12b -	UIC		The model is designed as a DEVS model with 1 input port for receiving the user requests and 2 output ports for redirecting the requests (already processed) to a specific instance of the PC component
	PC		The model is designed as a DEVS model with 5 input ports for receiving the requests (processed by some UIC component) and 3 output ports for redirecting the requests (already processed by itself) to a specific instance of the DAC component
	DAC		The model is designed as a DEVS model with 2 input ports for receiving the requests (processed by some PC component) and 1 output port for sending the final solutions of the initial user requests

On the other hand, the *number of couplings* K defined in DEVS implicit solutions can be defined as

$$K_{\text{DEVS}} = \text{number of EIC} + \text{number of IC} + \text{number of EOC}$$

where the *number of EIC*, *number of IC*, and *number of EOC* refer to the number of external input coupling, internal couplings, and external output couplings, respectively. However, we only consider internal couplings to capture the design of routing interactions.

Therefore, with $K_{\text{DEVS}} = \text{number of IC}$, the W/K relations for the case study are the same for both architectures: $W/K_{\text{DEVS},WA1} = W/K_{\text{DEVS},WA2} = 2$ (with $W_{\text{DEVS},WA1} = 40$ and $K_{\text{DEVS},WA1} = 20$ for *Web Architecture #1*, and $W_{\text{DEVS},WA2} = 32$ and $K_{\text{DEVS},WA2} = 16$ for *Web Architecture #2*). Such value is the one expected for “pre-wired” simulation models (as the one used in DEVS implicit solutions).




4.2.2 Building the simulation models using RDEVS formalism

Table 5 summarizes the set of simulation models required for modeling the web architectures depicted in Fig. 12 with RDEVS. In all cases, the model refers to the *Essential Model* that models the behavior of the architectural component.

Given that RDEVS employs separation of concerns between components and nodes, both scenarios use the same set of simulation models. Complementarily, as in the previous section, Appendix B illustrates the overall RDEVS representation for both scenarios using the set of simulation models defined as elements in Table 5.

In RDEVS solutions, each component is defined as an *Essential Model*. Then, for each *Essential Model* the modeler defines a set of *Routing Models* that improves the behavioral description by adding some specific routing policy. Therefore, in RDEVS solutions the *number of simulation models* Q depends on the influences among the *Essential Models* and *Routing Models*. If all *Routing Models* are defined over a single *Essential Model* (i.e. all *Nodes* employ the same *Com-*

Table 5 Basic simulation models required for modeling the web architectures scenarios using RDEVS

Web architecture	Architectural component	RDEVS model representation	Description
Web Architecture #1—Fig. 12a – and Web Architecture #2—Fig. 12b -	UIC		The model is designed as an Essential Model with 1 input port for receiving the user requests and 1 output port for sending the requests (already processed) to some PC component
	PC		The model is designed as an Essential Model with 1 input port for receiving the requests (processed by some UIC component) and 1 output port for sending the requests (already processed by itself) to some DAC component
	DAC		The model is designed as an Essential Model with 1 input port for receiving the requests (processed by some PC component) and 1 output port for sending the final solutions of the initial user requests

ponent), the value of Q will be $Q = N + 1$. Meanwhile, if each *Routing Model* is attached to some specific *Essential Model* (i.e. each *Node* refers to a different *Component*), the value of Q will be $Q = 2N$. Over such boundaries, an extra simulation model must be added to depict the scenario itself (that is, a *Network Model*). Therefore, in RDEVS solutions the *number of simulation models* Q is defined as:

$$N + 2 \leq Q_{RDEVS} \leq 2N + 1$$

To employ a similar approach to the one used for measuring DEVS implicit solutions, the *number of couplings* K is again defined as $K_{RDEVS} = \text{number of IC}$. However, the *number of IC* in RDEVS solutions is fixed as the set of all-to-all couplings required to develop a *Network Model*. Then, it is defined as

$$K_{RDEVS} = \text{number of IC}_{RDEVS} = N * (N - 1)$$

Following the case study, the measures obtained for RDEVS solutions are:

- For *Web Architecture #1*: Considering 3 *Essential Models*, 3 *Routing Models* for the UIC component, 4 *Routing Models* for the PC component, 2 *Routing Models* for the DAC component, and 1 *Network Model*, the *number of models* is $Q_{RDEVS,WA1} = 13$. Given that $N_{WA1} = 9$, the Q/N relation is $Q/N_{RDEVS,WA1} = 1.44$. Moreover, the W/K relation is $W/K_{RDEVS,WA1} = 0.1805$ (with $W_{RDEVS,WA1} = 13$ and $K_{RDEVS,WA1} = 72$).
- For *Web Architecture #2*: Following the approach detailed for *Web Architecture #1*, the *number of models* is $Q_{RDEVS,WA2} = 14$. Then, the Q/N relation is

$Q/N_{RDEVS,WA2} = 1.4$ (with $N_{WA2} = 10$). On the other hand, the W/K relation is $W/K_{RDEVS,WA2} = 0.1333$ (with $W_{RDEVS,WA2} = 12$ and $K_{RDEVS,WA2} = 90$).

4.3 Discussion

Table 6 summarizes the modeling effort for each routing scenario proposed in terms of the 3 tier architecture design. These values are obtained from the analysis performed in Sect. 4.2. The cases identified on the right side of the table refer to the valuation presented in Table 3.

Given that DEVS solutions employ fewer models than the RDEVS solution (because such solutions do not apply an appropriate separation of concerns), the RDEVS formalism shows an improvement of the Q/N relation. As Sect. 2.2 details, the use of an appropriate separation of concerns during the modeling task reduces the modeling effort. Moreover, when the W/K relation is considered along with the Q/N relation, the difference between DEVS and RDEVS modeling efforts for routing problems increases.

Even when the values are detailed for the case study, the modeling effort classification will remain the same when DEVS and RDEVS solutions were applied for modeling routing problems. DEVS solutions will always require modeling the component behavior along with the routing behavior. Instead, RDEVS solutions only require modeling the behavior of the components. Modeling only the component behaviors has two benefits for the modeler:

- Once the components are designed, they can be used to define any number of nodes: Building *Routing Model* instances does not involve an extra modeling effort for the modeler. Given that the *Essential Model* design

Table 6 Comparison between the modeling effort of DEVS and RDEVS solutions for the routing scenarios exemplified as 3 tier architectures

Routing scenario	Solution	Q/N	W/K	Modeling effort	
Web Architecture #1	DEVS	1.11	2	++	case #HIGH
	RDEVS	1.44	0.1805	-	case #LOW
Web Architecture #2	DEVS	1.1	2	++	case #HIGH
	RDEVS	1.4	0.1333	-	case #LOW

is used as a behavioral description of *Routing Model* instances, new *Routing Model* instances can be created by setting new routing policies over existent behaviors. The *Routing Model* instantiation only involves setting a routing policy over a predefined *Essential Model*. Then, such instantiation can be seen as a parametrization activity (not as a design task). For example, independently of the routing scenario, the number of components to be modeled in a 3 tier software architecture is 4. In the RDEVS solution for the case study, such models are *Essential Model* instances that depict the behavior of architectural components (Table 5). Instead, in DEVS solutions components are modeled along with nodes with aims to manage architectural functionality and routing as part of the models (e.g. the models detailed in Table 4). Therefore, the RDEVS models improve reusability and reduce the modeling effort.

- ii) *New routing scenarios can be defined using the same set of nodes with different routing policies:* Since *Routing Model* instances are attached to routing policies and the *Network Model* instance links several *Routing Models*, new scenarios can be defined by changing the routing policies definition. For example, if the web architecture scenario changes, the RDEVS solution detailed for the case study can be easily re-structured by adding/removing routing policies of specific *Routing Model* instances. Instead, an alternative web scenario in the DEVS solution will require not only the development of new models to add/remove nodes but also the modification of the existing ones. From this perspective, even when both solutions will require changes in the simulation models, the RDEVS solutions are more flexible and adaptable than the DEVS solution.

The improvement of the modeling effort in RDEVS is given by the modeling independence defined over *Components* and *Nodes*. Such modeling independence is reflected in the modeling levels attached to each formalism. In DEVS, the atomic DEVS defines the system behavior while the coupled DEVS defines the system structure. In RDEVS, the essential RDEVS defines the system domain behavior, the routing RDEVS defines the system routing behavior, and the network RDEVS defines the system structure. Then, by introducing a new modeling level (i.e., the system routing behavior), the RDEVS models provide an improvement in the modeling

effort when routing processes are defined over discrete-event models. Even when RDEVS uses more models, the primitives of conceptual modeling that act as a foundation for the modeling levels allows reusing the models by embedding domain functionality in the routing process definition. This fact is captured in the Q/N and W/K measures where the routing path definitions are transferred from DEVS explicit “pre-wired” connections to all-inclusive RDEVS simulation models. Such simulation models employ routing policies in the *Nodes* intending to define the overall *Routing Scenario*. Instead, DEVS models are strictly attached to the structure of the *Routing Scenario*. Hence, the RDEVS formalism reduces the modeling effort in routing simulation models by providing reusable solutions. The modeler only has to focus its modeling skills on the domain components without worrying about modeling the routing process.

To complete the analysis of *modeling complexity*, measures related to the *structural complexity* (Sect. 2.2) should be obtained. These measures cannot be obtained at design time. Measures such as costs of hierarchical structure, events encapsulation, and model’s embedding should be taken at execution time. These analyses will be carried out as future work. Moreover, the next steps will include the study of the performance and simulation times. However, in our experience, the simulation execution times for RDEVS models are not directly affected by the embedding of routing functionality. For example, when explicit routing solutions are built over DEVS models, the extra simulation model included for each *Node* (i.e., the *Handler*) needs to be added to the DEVS simulator hierarchy. Therefore, its introduction to handling the incoming and outgoing events requires the execution of transition functions and the propagation of input and output events. This feature affects the simulation times. In RDEVS-based solutions, such a feature is replaced with predefined behaviors for accepting incoming events and routing outgoing events. However, a deeper analysis will be conducted with the aim to provide suitable measures related to performance and simulation times.

4.4 Using RDEVS for the M&S of real-life scenarios

Regarding the use of DEVS and RDEVS to model real software architectures, in this section, we briefly explain how we

have applied such modeling in two case studies. These cases were obtained from Amazon Web Services.⁸

The first real case study analyzed is the Deals Engine Architecture of the Expedia Group Global.⁹ Figure 13 presents a representation of the architectural design. Following this representation, the architecture employs 7 components named *queue*, *elastic load balancer*, *loader*, *cache*, *mem. cache*, *web*, and *elastic search*. Over this set of components, a total of 15 nodes are defined. Some of these nodes share the same component. For example, 5 nodes share the architecture component named *elastic search*.

The second case study is depicted in Fig. 14. This case refers to the Multi-Region Resiliency of Netflix implemented with Amazon Route 53.¹⁰ The architecture uses 4 components (*flow logs*, *ec2*, *watchdog*, and *stream*) that structure 8 nodes.

Following the guidelines detailed in the previous section to define a simulation model for such a routing process situation, Table 7 shows the measures obtained for the modeling effort required when DEVS and RDEVS-based solutions are used to solve both cases. As in the 3-tier architecture case, it is important to denote that both DEVS and RDEVS strategies use the same model to describe component's behavior. Hence, as we have argued at the beginning, we use structure metrics (such as, the *number of ports* and *couplings*) along with the *number of models* and *nodes* to get modeling effort measures that allows us to compare situations where more complex models are used to define the same system.

For the DEVS-based solution of the first case, with a set of 7 components and 15 nodes, the value of N is 15, and the value of Q is 16 (i.e., $N + 1$ as defined in Sect. 4.2). Then, the value of Q/N is 1.0666. For the second case, with 4 components and 8 nodes, values are $N = 8$ and $Q = 9$ (again, $N + 1$). Here, the value of Q/N is 1.125. As discussed in Sect. 4.2, due to the “pre-wired” solution, the value of W/K is 2 for both cases.

On the other hand, for RDEVS-based solutions, measures are improved following the metrics defined in Sect. 4.2. In both cases, the *number of simulation models* Q is defined as $1 + N + \text{number of components}$. Moreover, the *number of couplings* K is calculated as $N*(N - 1)$. Finally, the *number of ports* W is obtained as $2N - (\text{number of external inputs} + \text{number of external outputs})$. Hence, for the first case with $N = 15$, $Q = 23$, $W = 24$, and $K = 210$, the ratios are $Q/N = 1.533$ and W/K is 0.1142. For the second case, with $N = 8$, $Q = 13$, $W = 11$, and $K = 56$, the ratios are $Q/N = 1.625$ and W/K is 0.1964.

⁸ URL: <https://aws.amazon.com/solutions/case-studies>.

⁹ AWS Expedia Case Study. Available at <https://aws.amazon.com/solutions/case-studies/expedia/> (accessed 10th June 2021).

¹⁰ Available at: <https://aws.amazon.com/solutions/case-studies/netflix/> (accessed 10th June 2021).

As in the generic 3-tier architecture case, the RDEVS solution improves the modeling effort for routing processes when, for example, the simulation model refers to software architectures. Due to the reuse of components in several nodes and the multiplied coupling definitions, the measures obtained in RDEVS-based solutions refer both to cases #LOW, while in DEVS-based solutions are #HIGH and #MEDIUM.

5 Conclusions and future work

Research and practice in systems modeling need approaches and tools that decrease the modeling effort due to the increasing (structural and behavioral) complexity of systems of systems. In this paper, we have presented the RDEVS formalism as a new DEVS extension that improves the design of discrete-event models that attempt to solve routing scenarios. RDEVS models are defined to manage the events flow using routing policies. Therefore, it structures the routing task through the use of routing policies executed at runtime as part of the simulation process.

We analyze our contribution from a conceptual modeling perspective to show how the core of RDEVS models is traced from the set of elements that compose a routing process definition. Moreover, we define and employ a set of metrics to estimate the modeling effort when RDEVS-based solutions are used instead of DEVS-based solutions. As future work, these metrics will be applied to evaluate the modeling effort in new scenarios (besides routing situations) obtained from the DEVS literature. This study will allow evaluating the need for new metrics to improve the modeling effort approach. Here, such modeling effort is valued in terms of the behavioral description designed for building the simulation model solution. This behavioral description is not seen as the traditional DEVS behavior model. Instead, it is focused on the modeling of an appropriate separation of concerns. By solving the routing process “inside” the structure of the simulation model, the RDEVS modeling task is less complex than its DEVS equivalent. That is, the modeling effort is reduced when RDEVS solutions are applied for routing scenarios. Therefore, the modeler can spend more time improving the achievement of the primary goal of the simulation model without worry about the routing process modeling (neither implicitly nor explicitly). Hence, the RDEVS formalism levels out the complexity of routing problem specifications to reduce the modeling effort. The use of RDEVS instead of DEVS is justified by the reduction of the modeling effort.

Even when RDEVS is an extension of DEVS, RDEVS models can be combined with DEVS models (atomic or coupled) to build powerful simulation models that exploit the benefits of each formalism according to the properties of the problem. Given that the core of RDEVS is DEVS, RDEVS models can be executed using DEVS simulators. That is, the

Fig. 13 Architecture representation of AWS Expedia Group

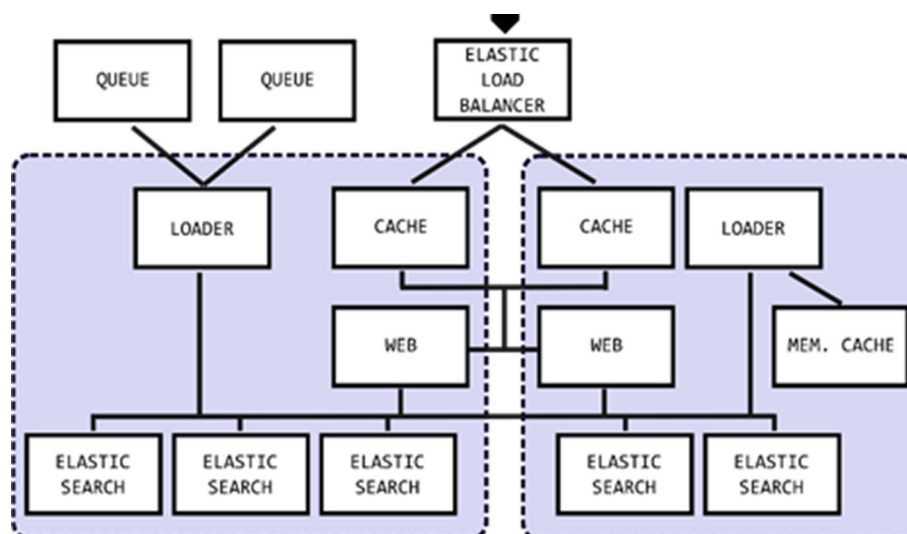


Fig. 14 Architecture representation of AWS Netflix Case Study

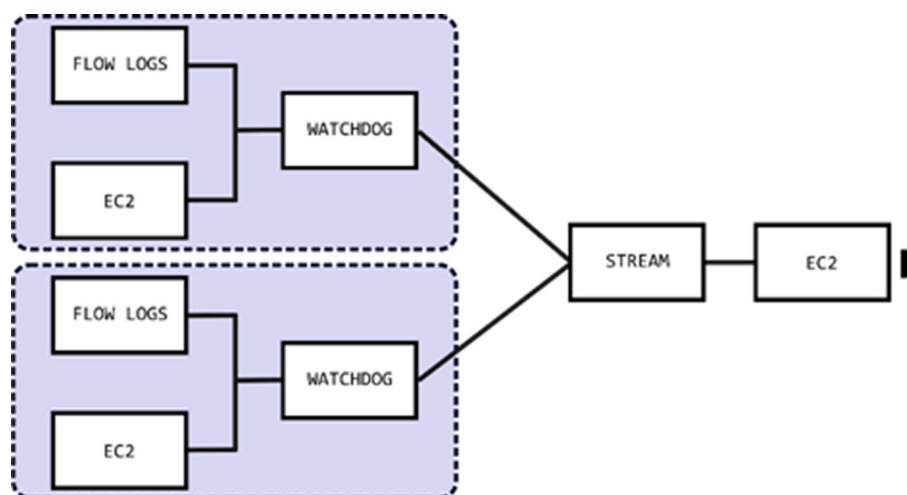


Table 7 Modeling effort measures for AWS Case Studies

Routing scenario	Solution	Q/N	W/K	Modeling effort	
AWS Expedia Group	DEVS	1.0666	2	++	case #HIGH
	RDEVS	1.533	0.1142	-	case #LOW
AWS Netflix Case Study	DEVS	1.125	2	++	case #MEDIUM
	RDEVS	1.625	0.1964	-	case #LOW

routing functionality of RDEVS models is encapsulated in the execution of the Routing model. The DEVS atomic model that describes the behavior of the node to be executed during the routing process simulation (i.e., the model identified as M in the Routing model definition) can be executed with any DEVS abstract simulator. Even when routing policies are hidden inside the definition of the models, their execution is embedded in the usual behavior of any DEVS model. Then, RDEVS and DEVS models are fully compatible during simulation execution (e.g. see [16]). However, new simulation algorithms could be designed to improve the simulation process according to the routing policies.

The RDEVS formalism provides complete and sound structural semantics for modeling routing processes over discrete-event models. *Reusability* and *flexibility* along with designs with *low coupling* and *high cohesion* are the main benefits of the formalism. By embedding the routing functionality into the models, it reduces the modeling effort required for building M&S solutions related to routing problems. So far, we have been used the final model specification for getting RDEVS implementations. The study on how the simulation scenario description can be used as a vehicle to get RDEVS implementations is part of our future work. Therefore, the next steps are oriented to developing M&S software

tools that facilitate the implementation of RDEVS models taking advantage of the modeling levels proposed in the formalism. Regarding the existing M&S software tools, we will study how RDEVS models can be introduced in these tools (e.g. as new software modules or libraries) to provide suitable implementations for M&S of general routing processes.

Appendix A

RDEVS closure under coupling

RDEVS network model to RDEVS routing model

The Network Model

$$N = \langle X, Y, D, \{R_d\}, \{I_d\}, \{Z_{i,d}\}, T_{in}, T_{out}, Select \rangle$$

where each $d \in D$ refers to a *RoutingModel* defined as R_d that is structured as

$$R_d = \langle \omega_d, E_d, M_d \rangle$$

defines an equivalent *Routing Model*

$$R = \langle \omega, E, M \rangle$$

in which:

- $\omega = (u, W, \delta_r) = (0, \phi, \delta_r) \mid \delta_r: S_M \rightarrow T_{OUT} \wedge T_{OUT} = \emptyset \equiv$ routing policy to be used in R. The equivalent model uses zero value as its identifier and an empty set of entities to represent the models from which input events are allowed.

$$s'_j = \begin{cases} \delta_{int,M,j}(s_j) & \text{if } j = 1 \\ \delta_{ext,M,j}(s_j, e_j + \tau_E(s), x_j) \text{ If } i^* \in I_j \wedge x_j \neq \emptyset \text{ with } x_j = Z_{i^*,j}(\lambda_{M,i^*}(s_{i^*})) & \\ s_j & \text{Otherwise} \end{cases}$$

These settings enable *Clause #3* of the external transition function detailed in the *Routing Model* definition. Then, R processes all input events that arrive. Moreover, by setting the routing function into an empty set value for any possible combination of T_{OUT} , all the output events of R will be sent everywhere (therefore, their processing depends on the receptor model configuration). Both behaviors are equivalent to the one expected of N since all input events that arrive at the *Network Model* are received but, also, all the output events created are sent.

- $E = \langle X_E, S_E, Y_E, \delta_{int,E}, \delta_{ext,E}, \lambda_E, \tau_E \rangle \equiv$ essential model embedded in R in which:

- o $X_E = X \equiv$ set of input events of E . As R is detailed as an equivalent model of N and, the *Routing Model* uses the inputs of E as part of its own inputs definition, the inputs of E are defined as equals to the inputs of N .
- p $S_E = \times_{i \in D} Q_i \mid Q_i = \{(s_i, e_i) \mid s_i \in S_{M,i}, 0 \leq e_i \leq \tau_{M,i}(s_i), \forall i \in D \equiv$ set of sequential states of E detailed as the product of the Q_i sets defined for each model that compose N (these are the *Routing Models*). Each Q_i is defined as an ordered pair that contains the state and the elapsed time of the R_i model.
- q $Y_E = Y \equiv$ set of output events of E . As in the case of input events, the output events of E are defined as equals to the output events of N .
- r $\delta_{int,E}(s) = s' \equiv$ internal transition function of E that modifies the state $s = (\dots, (s_j, e_j), \dots)$ to $s' = (\dots, (s'_j, e'_j), \dots)$ where $\{s, s'\} \in S_E$. Since the state of E is defined as a state combination of the models included in N , an internal transition of E may involve simultaneous internal transitions of multiple components. Then, considering that the imminent components (that is, the ones that must adjust its state) are collected according to the time value σ in a set structured as

$$IMM(s) = \{ i \in D \mid \sigma_i = \tau_{E,i}(s) \}$$

one model i^* must be selected to execute its internal transition. The N tie-breaking function can be used to get the i^* model. So, the imminent internal transition to be executed belongs to the R_{i^*} model where $i^* = Select(IMM(s))$. However, as a consequence of this transition, all external transitions of the components influenced by R_{i^*} must be executed. So, the final state transformation from $s = (\dots, (s_j, e_j), \dots)$ to $s' = (\dots, (s'_j, e'_j), \dots)$ is defined by

$$e'_j = \begin{cases} 0 & \text{if } (j = i^*) \vee (i^* \in I_j \wedge x_j \neq \emptyset) \\ e_j + \tau_E(s) & \text{otherwise} \end{cases}$$

- s $\delta_{ext,E}(s, e, x) = s' \equiv$ external transition function of E that modifies the set of state pairs that refers to the R_i models linked to the inputs of N . Given that the state of E is defined as S_E , the states $\{s, s'\} \in S_E$ where $s = (\dots, (s_i, e_i), \dots)$ and $s' = (\dots, (s'_i, e'_i), \dots)$. Considering that components are collected in a set $C = \{ i \in D \mid N \in I_i \wedge x_i \neq \emptyset \}$, then

$$s_i^* = \delta_{ext,M,i}(s_i, e_i + e, x_i) \text{ with } x_i = Z_{N,i}(x), \forall i \in C$$

so the state transformation is defined as

$$(s'_i, e'_i) = \begin{cases} (s_i^*, 0) & \text{if } (N \in I_i \wedge x_i \neq \emptyset) \\ (s_d, e_d + e) & \text{otherwise} \end{cases}$$

- t $\lambda_E(s): S_E \rightarrow Y_E \cup \emptyset \equiv$ output function of E that generates an output event if and only if the model that is going to execute its internal transition (that is, i^* model) is linked to the outputs of N . The function is defined as

$$\lambda_E(s) = \begin{cases} Z_{i^*, N}(\lambda_{M, i^*}(s_{i^*})) & \text{if } N \in I_{i^*} \\ \emptyset & \text{otherwise} \end{cases}$$

- u $\tau_E: S_E \rightarrow R_{0, \infty}^+ \equiv$ time advance function of E that select the most imminent event time of all the components (that is, the routing models) included in N (i.e. finding the smallest remaining time σ until the internal transition of all the simulation models included in N). The function is defined as

$$\tau_E(s) = \min \{ \sigma_i = \sigma_{M, i}(s_i) - e_i \mid i \in D \}$$

- $M = \langle X_M, S_M, Y_M, \delta_{\text{int}, M}, \delta_{\text{ext}, M}, \lambda_M, \tau_M \rangle \equiv$ DEVS atomic model that specifies the routing process of R . Given that the description of M is defined in the routing model definition and, considering that its specification uses some of the components defined in E , no considerations are required to get the equivalent model of N .

RDEVS routing model to RDEVS essential model

The *Routing Model* specification includes two DEVS models defined as E and M . To define an *Essential Model* that acts as an equivalent model of a *Routing Model* description, it is important to understand the difference between both models. While E determines the *Component* to be used as part of the *Node* (that is, the *Essential Model* that describes the behavior of the *Routing Model*), M defines the executable simulation model over which the routing process takes place. Then, the equivalence proof tries to find a *Component* with the same

behavior that a *Node*. Moreover, it can use the *Node* description as part of the *Component* specification since both models belong to the same type (DEVS atomic model).

Then, the *Routing Model* described by the structure

$$R = \langle \omega_R, E_R, M_R \rangle$$

with $M_R = \langle X_{M,R}, S_{M,R}, Y_{M,R}, \delta_{\text{int}, M,R}, \delta_{\text{ext}, M,R}, \lambda_{M,R}, \tau_{M,R} \rangle$, can be described as an equivalent *Essential Model* structured as

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \tau \rangle$$

in which $X = X_{M,R}, S = S_{M,R}, Y = Y_{M,R}, \delta_{\text{int}} = \delta_{\text{int}, M,R}, \delta_{\text{ext}} = \delta_{\text{ext}, M,R}, \lambda = \lambda_{M,R}$ and $\tau = \tau_{M,R}$. Following this equivalence, each component of M_R (that is, the executable model of the routing model description) is directly mapped to a new model that defines an *Essential Model* that maintains the desired behavior of the *Routing Model*.

Appendix B

Representation of web architectures as discrete-event simulation models

DEVS representation

Figures 15 and 16 show the representation of the web-based architectures depicted in Fig. 12 as DEVS models. Each box included in the figures refer to a DEVS model detailed in Table 4. In Fig. 15, we use the models defined in the first row of Table 5 (i.e., the ones designed for Fig. 12a). Instead, in Fig. 16, we use the models defined in the second row of the table (i.e., the DEVS models detailed for the architecture depicted in Fig. 12b).

RDEVS representation

Figures 17 and 18 show the representation of the web-based architectures depicted in Fig. 12 as RDEVS models. Each box included in the figures refer to a RDEVS model detailed in Table 5. In both cases, the same set of models is used.

Fig. 15 Mapping the Web Architecture #1 into DEVS models detailed in Table 4

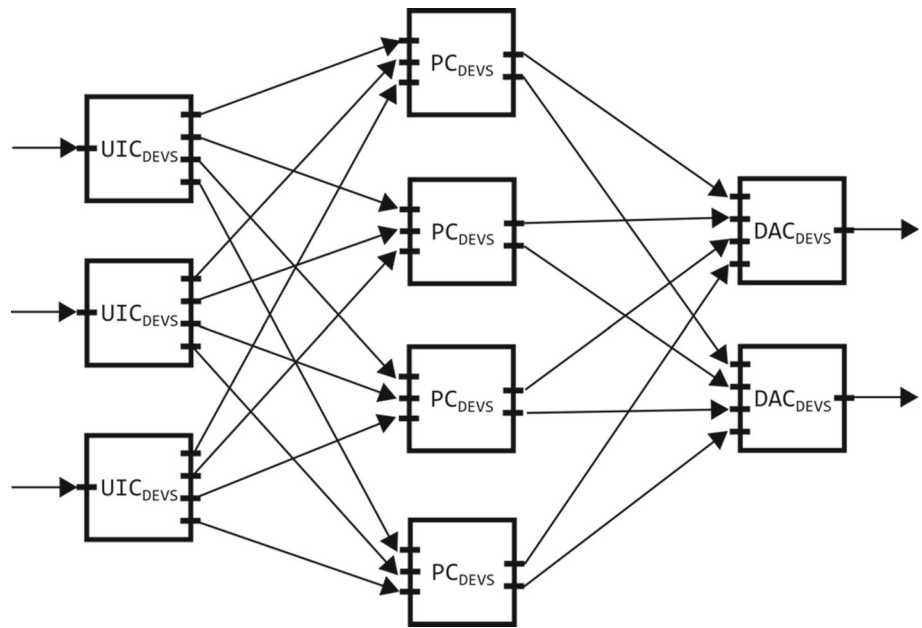


Fig. 16 Mapping the Web Architecture #2 into DEVS models detailed in Table 4

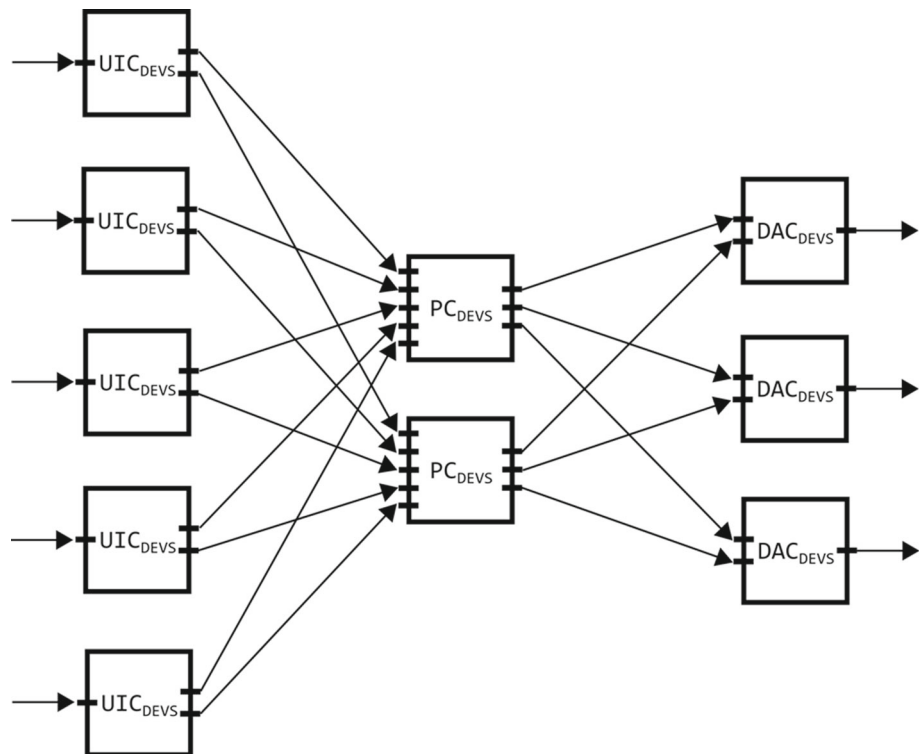


Fig. 17 Mapping the Web Architecture #1 into RDEVS models detailed in Table 5

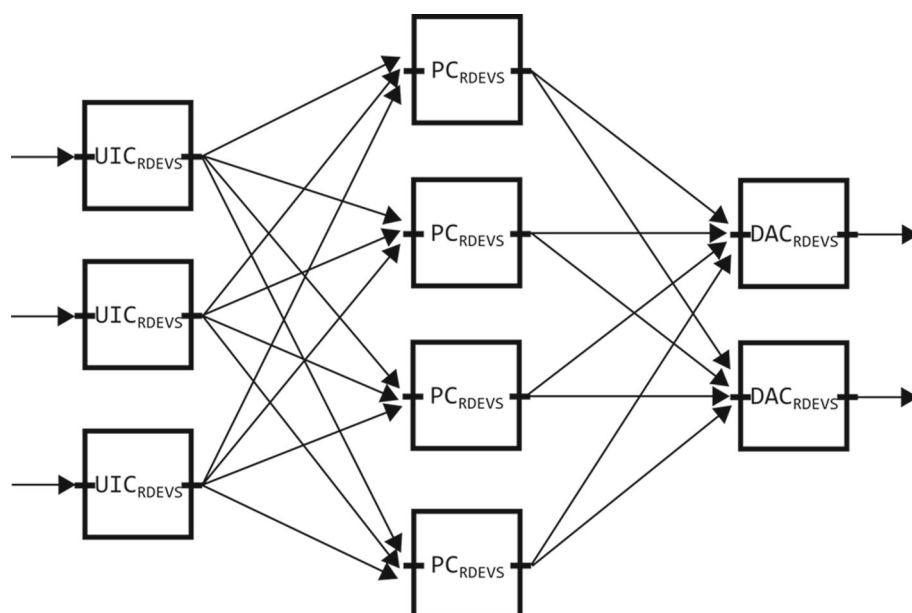
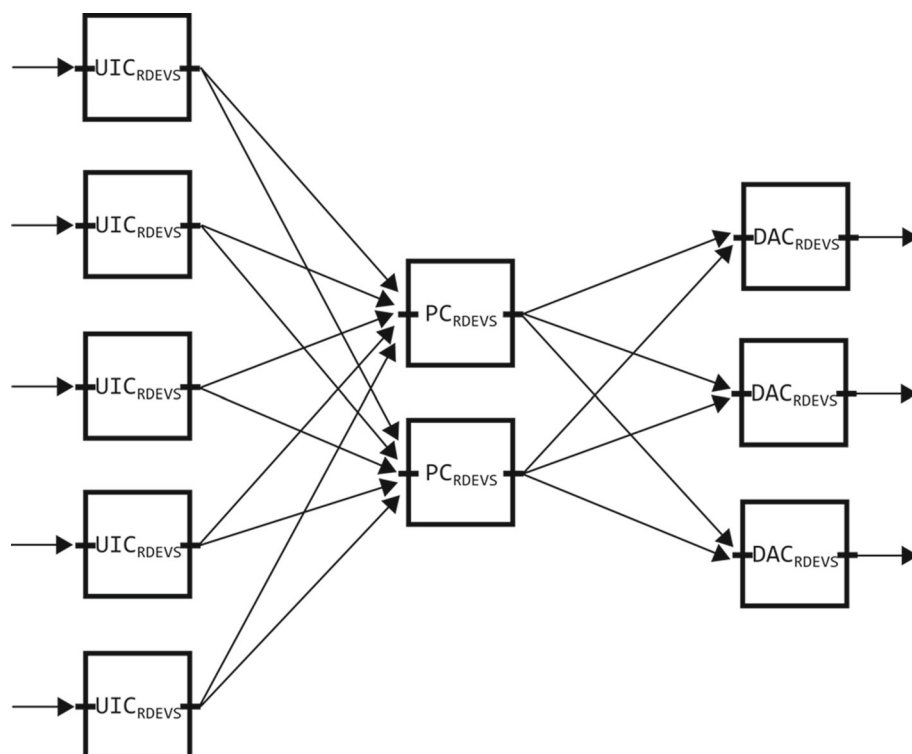


Fig. 18 Mapping the Web Architecture #2 into RDEVS models detailed in Table 5



References

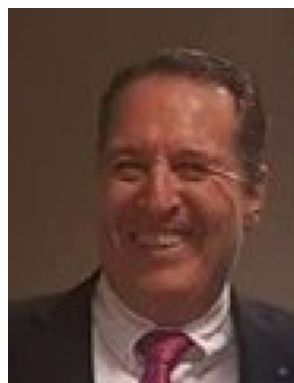
1. Zeigler, B., Muzy, A., Kofman, E.: Theory of modelling and simulation: discrete event and iterative system computational foundations, 3rd edn. Academic Press, Cambridge (2018)
2. Y.J. Kim, J.Y. Yang, Y.M. Kim, J. Lee, C. Choi, Modeling Behavior of Mobile Application Using Discrete Event System Formalism, In: Proceeding of the 2016 Asian Simulation Conference, 2016, pp. 40–48. https://doi.org/10.1007/978-981-10-2158-9_4.
3. G. Wainer, M. Etemad, B. Kazi, Modeling Coordinated Multipoint with a dynamic Coordination Station in LTE-A mobile networks, In: Proceeding of the 2017 IEEE International Conference on Networking, Sensing and Control, 2017, pp. 807–812. <https://doi.org/10.1109/ICNSC.2017.8000194>.
4. S. Gholami, H. Sarjoughian, G. Godding, D. Peters, V. Chang, Developing Composed Simulation and Optimization Models using Actual Supply-Demand Network Datasets, In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 2510–2521. <https://doi.org/10.1109/WSC.2014.7020095>.

5. Uhrmacher, A., Degenring, D., Zeigler, B.: MultiLevel discrete-event modeling in systems biology. *Trans. Comput. Syst. Biol.* **1**, 66–89 (2004). https://doi.org/10.1007/978-3-540-32126-2_6
6. A. Uhrmacher, R. Ewald, M. John, C. Maus, M. Jeschke, S. Biermann, Combining micro and macro-modeling in DEVS for computational biology, In: Proceedings of the 2007 Winter Simulation Conference, 2007, pp. 871–880. <https://doi.org/10.1109/WSC.2007.4419683>.
7. M. Moallemi, G. Wainer, A. Awad, D.A. Tall, Application of RT-DEVS in Military, In: Proceedings of the 2010 Spring Simulation Multiconference, 2010, pp. 29–36. <https://doi.org/10.1145/1878537.1878568>.
8. Bogado, V.S., Gonnet, S., Leone, H.: Modeling and simulation of software architecture in discrete event system specification for quality evaluation. *Simul.* **90**, 290–319 (2014). <https://doi.org/10.1177/0037549713518586>
9. J.L. Risco-Martín, S. Mittal, J.C. Fabero, P. Malagón, J.L. Ayala, Real-time Hardware/Software co-design using Devs-Based Transparent M&S Framework, In: Proceedings of the 2016 Summer Computer Simulation Conference, 2016, 45–52. <https://doi.org/10.22360/SummerSim.2016.SCSC.053>.
10. Blas, M.J., Gonnet, S., Leone, H.: Modeling user temporal behaviors using hybrid simulation models. *IEEE Lat. Am. Trans.* **15**, 341–348 (2017). <https://doi.org/10.1109/TLA.2017.7854631>
11. Van Tendeloo, Y., Vangheluwe, H.: An evaluation of DEVS simulation tools. *Simul.* **93**, 103–121 (2017). <https://doi.org/10.1177/0037549716678330>
12. Robinson, S.: Conceptual modelling for simulation Part I: definition and requirements. *J. Op. Res. Soc.* **59**, 278–290 (2008). <https://doi.org/10.1057/palgrave.jors.2602368>
13. D. Cetinkaya, A. Verbraeck, M.D. Seck, A metamodel and a DEVS implementation for component based hierarchical simulation modeling, In: Proceedings of the 2010 Spring Simulation Multiconference, 2010, p. 170. <https://doi.org/10.1145/1878537.1878714>.
14. Santucci, J.-F., Capocchi, L., Zeigler, B.P.: System entity structure extension to integrate abstraction hierarchies and time granularity into DEVS modeling and simulation. *Simul.* **92**, 747–769 (2016). <https://doi.org/10.1177/0037549716657168>
15. M.J. 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. <https://doi.org/10.1109/CLEI.2016.7833339>.
16. Blas, M.J., Leone, H.P., Gonnet, S.M.: Modeling and simulation framework for quality estimation of web applications through architecture evaluation. *SN Appl. Sci.* **2**, 374 (2020). <https://doi.org/10.1007/s42452-020-2171-z>
17. Alexander, C.: A pattern language: towns, buildings, construction. Oxford University Press, New York (1977)
18. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design patterns: Abstraction and reuse of object-oriented design, In: European Conference on Object-Oriented Programming, Springer, Berlin, 1993, pp. 406–431. https://doi.org/10.1007/978-3-642-48354-7_15.
19. D. Cetinkaya, A. Verbraeck, M.D. Seck, MDD4MS: a model driven development framework for modeling and simulation, In: Proceedings of the 2011 summer computer simulation conference, 2011, pp. 113–121.
20. Wainer, G., Giambiasi, N.: N-dimensional Cell-DEVS models. *Discret. Event Dyn. Syst.* **12**, 135–157 (2002). <https://doi.org/10.1023/A:1014536803451>
21. Barros, F.J.: Modeling formalisms for dynamic structure systems. *ACM Trans. Model. Comput. Simul.* **7**, 501–515 (1997). <https://doi.org/10.1145/268403.268423>
22. Y. Kwon, H. Park, S. Jung, T. Kim, Fuzzy-DEVS Formalism: Concepts, Realization and Application, In: Proceedings of the 1996 Conference on Artificial Intelligence, Simulation and Planning In High Autonomy Systems, 1996, pp. 227–234.
23. Hamri, M., Giambiasi, N., Frydman, C.: Min–Max-DEVS modeling and simulation. *Simul. Model. Pract. Theory* **14**, 909–929 (2006). <https://doi.org/10.1016/j.simpat.2006.05.004>
24. Steniger, A., Uhrmacher, A.: Intensional coupling in variable structure models: an exploration based on multi-level DEVS. *ACM Trans. Model. Comput. Simul.* **26**(2), 1–27 (2016)
25. A.C. Chow, B. Zeigler. Parallel DEVS: a Parallel, Hierarchical, Modular Modeling Formalism, In: Proceedings of the 1994 Winter Simulation Conference, 1994, pp. 716–722. <https://doi.org/10.1109/WSC.1994.717419>.
26. Hong, J.S., Song, H.S., Kim, T.G., Park, K.H.: A real-time discrete event system specification formalism for seamless real-time software development. *Discret. Event Dyn. Syst.* **7**, 355–375 (1997). <https://doi.org/10.1023/A:1008262409521>
27. Bergero, F., Kofman, E.: A Vectorial DEVS Extension for Large Scale System Modeling and Parallel Simulation. *Simul.* **90**, 522–546 (2014). <https://doi.org/10.1177/2F0037549714529833>
28. M.J. Blas, S. Gonnet, H. Leone, B. Zeigler, A conceptual framework to classify the extensions of DEVS formalism as variants and subclasses, In: Proceedings of the 2018 Winter Simulation Conference, 2018, pp. 560–571. <https://doi.org/10.1109/WSC.2018.8632265>.
29. Liu, Q., Wainer, G.: Parallel environment for DEVS and Cell-DEVS models. *Simul.* **83**, 449–471 (2007). <https://doi.org/10.1177/2F0037549707085084>
30. M.J. Blas, S. Gonnet, H. Leone, 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, 2017, pp. 774–785. <https://doi.org/10.1109/WSC.2017.8247831>.
31. Rumbaugh, J., Jacobson, I., Booch, G.: The unified modeling language reference manual, 2nd edn. Pearson Higher Education, New Jersey (2004)
32. Batty, M., Torrens, P.M.: Modelling complexity: the limits to prediction. *Cybergeo: Eur. J. Geogr.* (2001). <https://doi.org/10.4000/cybergeo.1035>
33. Capocchi, L., Santucci, J.-F., Pawletta, T., Folkerts, H., Zeigler, B.P.: Discrete-event simulation model generation based on activity metrics. *Simul. Model. Pract. Theory* **103**, 102122 (2020). <https://doi.org/10.1016/j.simpat.2020.102122>
34. Brooks, R.J., Tobias, A.M.: Choosing the best model: Level of detail, complexity, and model performance. *Math. Comput. Model.* **24**(4), 1–14 (1996). [https://doi.org/10.1016/0895-7177\(96\)00103-3](https://doi.org/10.1016/0895-7177(96)00103-3)
35. L. Schruben, E. Yucesan, Complexity of simulation models a graph theoretic approach, In: Proceedings of 1993 Winter Simulation Conference, 1993, pp. 641–649. <https://doi.org/10.1109/WSC.1993.718302>.
36. Robinson, S., Nance, R.E., Paul, R.J., Pidd, M., Taylor, S.J.: Simulation model reuse: definitions, benefits and obstacles. *Simul. Model. Pract. Theory* **12**, 479–494 (2004). <https://doi.org/10.1016/j.simpat.2003.11.006>
37. Farooq, U., Wainer, G., Balya, B.: DEVS modeling of mobile wireless ad hoc networks. *Simul. Model. Pract. Theory* **15**, 285–314 (2007). <https://doi.org/10.1016/j.simpat.2006.11.011>
38. 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 - DEVS Integrative, 2014, article 20.
39. 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, 2017, article 9. <https://doi.org/10.22360/springsim.2017.mod4sim.014>.

40. I. Dávid, H. Vangheluwe, Y. Van Tendeloo, Translating engineering workflow models to DEVS for performance evaluation, In: Proceedings of the 2018 Winter Simulation Conference, 2018, pp. 616–627. <https://doi.org/10.1109/WSC.2018.8632470>.
41. A. Alshareef, H. Sarjoughian, Metamodeling activities for hierarchical component-based models, In: Proceedings of the Theory of Modeling and Simulation Symposium, 2019, article 2. <https://doi.org/10.23919/SpringSim.2019.8732854>.
42. ISO/IEC TR 19759:2015, Software Engineering - Guide to the Software Engineering Body of Knowledge (SWEBOK), 2015.
43. M. Tavanpour, J. Mikhail, G. Wainer, G. Boudreau, The case for DEVS in networking M&S: upload user collaboration in mobile networks using coordinated multipoint, In: Proceedings of the Theory of Modeling and Simulation Symposium, 2017, article 8.
44. Grimm, V., Revilla, E., Berger, U., Jeltsch, F., Mooij, W.M., Railsback, S.F., DeAngelis, D.L.: Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *Sci.* **310**(5750), 987–991 (2005). <https://doi.org/10.1126/science.1116681>
45. B. Zeigler, Closure Under Coupling: Concept, Proofs, DEVS Recent Examples, In: Proceedings of the 2018 Spring Simulation Multi-conference, 2018. <https://doi.org/https://doi.org/10.1145/3213187.3213194>.
46. M. Hitz, B. Montazeri, Measuring coupling and cohesion in object-oriented systems, In: Proceedings of International Symposium on Applied Corporate Computing, pp. 25–27, 1995.
47. Offutt, A.J., Harrold, M.J., Kolte, P.: A software metric system for module coupling. *J. Syst. Softw.* **20**, 295–308 (1993). [https://doi.org/10.1016/0164-1212\(93\)90072-6](https://doi.org/10.1016/0164-1212(93)90072-6)
48. Bieman, J.M., Kang, B.K.: Measuring design-level cohesion. *IEEE Trans. Software Eng.* **24**, 111–124 (1998). <https://doi.org/10.1109/32.666825>
49. Page-Jones, M.: The practical guide to structured systems design. Yourdon Press, New York (1980)
50. Shaw, M., Garlan, D.: Software architecture: perspectives on an emerging discipline. Prentice Hall, Englewood Cliffs (1996)
51. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud computing patterns: fundamentals to design, build, and manage cloud applications. Springer, Berlin (2014)



Silvio Gonnet received his PhD degree in Engineering from Universidad Nacional del Litoral in 2003. He currently holds a Researcher position at the National Scientific and Technical Research Council (CONICET). His research interests include conceptual modeling, ontology engineering, and discrete-event modeling and simulation.



Horacio Leone is Professor of the Department of Information Systems Engineering at the Facultad Regional Santa Fe (FRSF), Universidad Tecnológica Nacional (UTN) and Principal Investigator of CONICET at the Institute of Development and Design - INGAR (CONICET- UTN). Dr. Leone received his Ph.D. in Chemical Engineering from the Universidad Nacional del Litoral in 1986. Also, he was a postdoctoral fellow at the Laboratory for Intelligent Systems in Process Engineering (MIT). His research interests include software systems simulation and ontology applications to manufacturing systems.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



María Julia Blas is a Postdoctoral Fellow at Instituto de Desarrollo y Diseño INGAR (Argentina) and an Assistant Professor at Universidad Tecnológica Nacional – Facultad Regional Santa Fe (UTN-FRSF). She received her Ph.D. degree in Engineering from Universidad Tecnológica Nacional in 2019. Her research interests include discrete-event M&S.