A CONCEPTUAL FRAMEWORK TO CLASSIFY THE EXTENSIONS OF DEVS FORMALISM AS VARIANTS AND SUBCLASSES

María J. Blas
Silvio M. Gonnet
Horacio P. Leone

Bernard P. Zeigler

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

University of Arizona
RTSync Corp.
12500 Park Potomac Ave.
Potomac, MD, 20854, USA

ABSTRACT

The Discrete Event System Specification (DEVS) is a general modeling formalism with sound semantics founded on a system theoretic basis. It can be used as a base for the development of specialized modeling formalisms. Usually, the extensions of DEVS expand the classes of systems models that can be represented in DEVS. However, with a growing number in new variants of DEVS and an increasing number of problems to be solved using discrete simulation techniques, it is necessary to define the relations among different approaches. This paper presents a conceptual modeling perspective applied to DEVS extensions that structure a framework over the traditional modeling and simulation approach. The framework provides a multilevel structure to analyze the features required for each extension type. Two main types of extensions are identified: variants and subclasses. In order to illustrate the proposed guidelines, the Routed DEVS formalism is presented as example of the subclass type.

1   INTRODUCTION

The Discrete Event System Specification (DEVS) is a general modeling formalism with sound semantics founded on a system theoretic basis (Zeigler and Vahie 1993). This formalism embodies a set of concepts related to systems theory and modeling in order to describe discrete event models. It provides a general methodology for hierarchical construction of reusable models in a modular way.

Wainer and Mosterman (2010) propose that the core of DEVS is composed of: i) a set of *basic concepts* defined in terms of the Discrete Event Systems Specification (DEVS), the DEVS Simulation (performed by a DEVS Simulator that implements an Abstract DEVS Simulator) and the System Entity Structure (which is a formal structure with a set of axioms that provides valid models; ii) a *modeling and simulation (M&S) framework* defined as an ontology of the modeling and simulation domain expressed in terms of entities and their relationships; iii) a *systems-theory basis* used to formulate the framework entities in terms of system specifications, and the framework relations in terms of the morphisms among system specifications. However, in the past years, advances have been made radiating out from the DEVS core (Zeigler et al. 2000).

In this context, DEVS can be used as a base for the development of specialized modeling formalisms for specific domains (Wainer and Mosterman 2010). Several researchers have proposed extensions and new formalisms derived from DEVS in order to solve different scenarios. As problems' complexity increase, new mechanisms are required to deal with them. Furthermore, specialized DEVS may be useful for a plethora of applications (Wainer and Mosterman 2010). Usually, the extensions of the DEVS formalism expand the classes of systems models that can be represented in DEVS (Zeigler et al. 2000). This statement is true (at least) for Dynamic Structure DEVS (DSDEVS) (Barros 1997), Symbolic DEVS

(Zeigler and Chi 1992), Real Time DEVS (RT-DEVS) (Hong et al. 1997) and Fuzzy DEVS (Kwon 1996). However, with a growing number in new variants of DEVS and an increasing number of problems to be solved using discrete simulation techniques, it is necessary to define the relations among different approaches. Each extension of DEVS suggests a direction in which simulation modeling in general, and DEVS formalism in particular, can be enhanced. However, in order to ensure this dependency, it is necessary to analyze the impact of the new extension over the original formalism.

Frequently, the conceptual modeling field is related with the set of mechanisms used to study a certain domain. From a systemic point of view, conceptual modeling can be defined as "the process of abstracting a model from a real or proposed system" (Robinson et al. 2010). Then, its main purpose is to elicit the conceptual schema of a specific system (Olivé 2007). Several authors have used these types of models in order to illustrate different simulation approaches (Yilmaz and Oren 2004; Robinson 2006; Furian et al. 2015; Robinson et al. 2015; Abdelmegid et al. 2017). All these approaches deal with the conceptual modeling field as a problem related to the modeling phase when building a simulation model. However, from a broad point of view, conceptual modeling can be used to support ontological designs for several domains. In this context, conceptual modeling is concerned with identifying, analyzing and describing the essential concepts and constraints of a universe of discourse with the help of a (diagrammatic) modeling language that is based on a set of basic modeling concepts (Guizzardi and Halpin 2008).

This paper presents a conceptual modeling perspective applied to the domain of DEVS formalism that structure a set of entities as a conceptual framework applicable over the M&S framework in order to illustrate the main features of DEVS extensions. The framework proposed defines the set of main concepts, relationships and properties required to classify the extensions of DEVS applying two dimensions: *system* and *problem*. Both dimensions are used to characterize different types of DEVS extensions. In order to illustrate the proposed guidelines, the Routed DEVS formalism (Blas et al. 2017) is used as case study to represent the *DEVS subclass type*. The conceptual framework designed also encourages the DEVS bus approach in which simulators for models expressed in various event-based formalisms interoperate under the control of a standard DEVS coordinator (Zeigler et al. 2000). The set of properties defined as part of the framework can be used as a vehicle to guarantee the consistency between different extensions of DEVS in order to ensure the applicability of the simulators.

The remainder of this paper is structured as follows. Section II describes the foundations used to design the framework along with the main abstractions identified as part of the DEVS extensions domain. Section III presents the conceptual framework designed to structure the extensions of DEVS in order to study the mandatory properties required for each extension type. Section IV shows how a the *DEVS subclass* type must be interpreted using Routed DEVS as a case study. Finally, Section V is devoted to conclusions and future work.

2 FOUNDATIONS

The M&S framework defines a set of entities and their relationships in order to illustrate the M&S domain (Figure 1). The basic entities identified in the framework are *source system*, *model*, *simulator* and *experimental frame*. The *source system* is the real or virtual environment to be modeled. The *model* describes the set of instructions for generating data comparable to the data observable in the real system. The *simulator* refers to the computational system that executes the instructions detailed in the model. Finally, the *experimental frame* represents the conditions under which the system is observed or experimented with. Following this abstraction, each component is defined as an individual element in order to keep the entities independence.

In order to link the entities properly, two main relationships are defined: *modeling* and *simulation*. The *modeling relation* determines when a model can be said to be a valid representation of a source system within an experimental frame. The *simulation relation* specifies what constitutes a correct simulation of a model by a simulator.
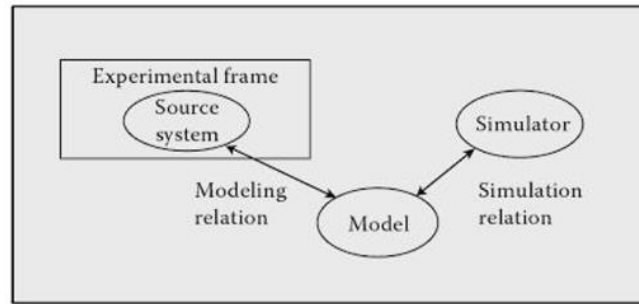
Figure 1: M&S framework proposed by Zeigler and Sarjoughian (2003).

Based on this framework, the basic issues and problems encountered in performing M&S activities can be better understood and coherent solutions developed (Zeigler et al. 2000). Keeping the components as independent entities provides several benefits, such as: i) the same model can be executed by different simulators, ii) several experiments can be executed to study different situations, and iii) different models (or versions of the same model) can be tested under the same experiment using an unique simulator. Also, the strict separation of models and simulators facilitates the development of alternative algorithms to dramatically speed up the simulation of the models (Wainer and Mosterman 2010). Furthermore, since the M&S framework is defined as an ontology of the M&S domain, it can be applied to different areas or situations. Each application should be considered a M&S framework instance in which all the properties of the original model should be valid. Commonly, each instantiation is attached to some basic system formalism.

When the M&S framework is applied to a discrete event system described using DEVS formalism, an implicit M&S framework instance is specified to solve the problem. This instance is DEVS-specific because it refers to a DEVS model that must be executed by a DEVS simulator with an experimental frame compatible with DEVS. Moreover, given that the instance follows the M&S framework definition, it keeps the independence property defined in the domain entities of the original approach. Then, i), ii) and iii) are valid statements for the DEVS-specific instance of the M&S framework. However, when the framework instance becomes DEVS-specific, the statements related to its definition also became DEVS-specific. For example, a specific DEVS simulator entity can only be replaced by a similar one without changing the other entities instance. Although the M&S instance was created for a basic system formalism (that is, DEVS), the same *formalism-specific* property maintains for DEVS extensions.

Frequently, DEVS extensions are related to specific simulators or are partially incompatible with the execution of traditional DEVS formalism approaches. Then, a model built with a DEVS extension is not always replaceable with a DEVS model and, reciprocally, a DEVS model does not always have an equivalent in a DEVS extension approach. This lack of equivalence is given by the DEVS extension type. Although these extensions expand the classes of systems models that can be represented in DEVS, the perspective from which each approach is developed influences the formalisms equivalence property.

Following the taxonomy presented by Zeigler et al. (2000), a separation of concerns can be depicted in order to classify DEVS extensions using two different criteria over the original DEVS formalism: *types of systems* and *types of problems*. Moreover, two analysis dimensions can be depicted using these criteria in order to define a set of desired properties related to the extensions of the formalism. In this context, the *systems dimension* refers to the formalism capability to represent new classes of dynamic systems applying discrete-event models. On the other hand, the *problems dimension* refers to the formalism capability to allow building less complex models when solving specific kinds of situations. Figure 2 represents these two dimensions using a quadrangular perspective depicted over two axes. The horizontal axis represents the problems dimension used to contextualize the specializations of DEVS developed to solve different types of problems. The *subclasses of DEVS* are represented over this axis because they refer to the subset of DEVS extensions in which the alternative formalism improves the solution of the

simulation problem applying DEVS models in a meaningful way. Then, these extensions are *highly domain-dependent* because improve some domain problem representation. On the other hand, the vertical axis refers to the systems dimension used to specify the different types of systems models that can be defined from DEVS formalism. The *variants of DEVS* are represented over this axis because they refer to the subset of DEVS extensions in which the alternative formalism models a new type of system that, previously, could not be modeled with the original formalism. In this case, the extensions of DEVS are *context-exclusive* from the system to be modeled using the formalism because they allows representing some new type of essential system. Finally, the *hybrid DEVS* section represents a mixed approach between both dimensions.

For each type of extension identified in Figure 2, an example is included in order to illustrate the proposed classes. Since Classic DEVS and Parallel DEVS (P-DEVS) are most popular formalisms and considering that both built the foundations for most extensions, both formalism can be placed at the core class (that is, *DEVS*). The Routed DEVS extension is classified as *subclass of DEVS* (this is formally presented in Section 4), while the DSDEVS is proposed as *variant of DEVS* because it allows to model systems that exhibit dynamic structures. Finally, Cell-DEVS is proposed as example of the *hybrid DEVS* class because it provides an extension useful to model a specific type of problems for a specific type of systems.
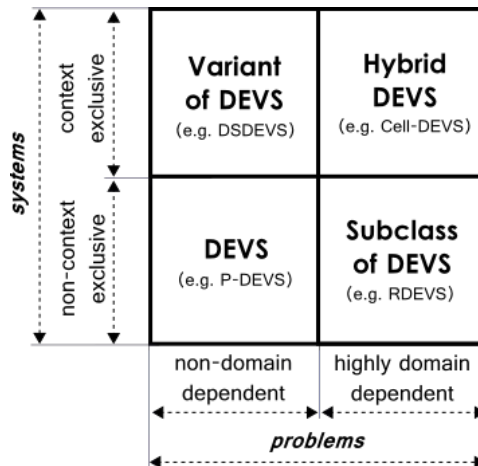

Figure 2: Types of DEVS extensions.

This contextualization gives a novel approach to classify the extensions of DEVS. It provides a mechanism to study each extension type in order to make a comparison between several approaches. Then, the extensions of DEVS can be interpreted in terms of similarities and dissimilarities by applying the dimensions proposed with aims to understand their main features.

Furthermore, the dimensions used to define the types of DEVS extensions are directly linked to the entities identified as part of the M&S framework (Figure 3). The *types of systems* concept influences the *source system entity* because it gives the possibility of modeling new types of sources with discrete event formalism. Then, a broad set of dynamic systems is available to be used as part of specific instances of the M&S framework. Likewise, the *types of problems* concept impacts on the *model entity* because it allows solving a common modeling situation with an improved specification. Then, a broad set of models can be designed for the same source system applying several specifications based on the same formalism.

As Figure 3 shows, each dimension is attached to an entity involved in the *modeling relation*. This feature indicates that, in both cases, the dimensions proposed can be understood as modeling factors that influence the relation. Given that these dimensions are used to characterize the extensions of DEVS, the types identified as part of the approach can be used as templates to interpret how DEVS extensions improve the *modeling relation* (that is basically representing the *modeling task*). Formally, each type of

extension influences some specific component of the relation; that is: i) A *subclass of DEVS* affects the *model entity* because it **helps to develop less complex models for specific problems** (decreasing the complexity of the modeling task). ii) A *variant of DEVS* promotes the **possibility to develop simulation models for new kinds of** *source systems entities* (giving a representation of the modeling relation in new environments). iii) As *hybrid DEVS* is a combination of the other extension types, it influences the modeling relation combining the properties described in i) and ii). Then, each type of extension acts as a vehicle to set up the modeling relation through its components.
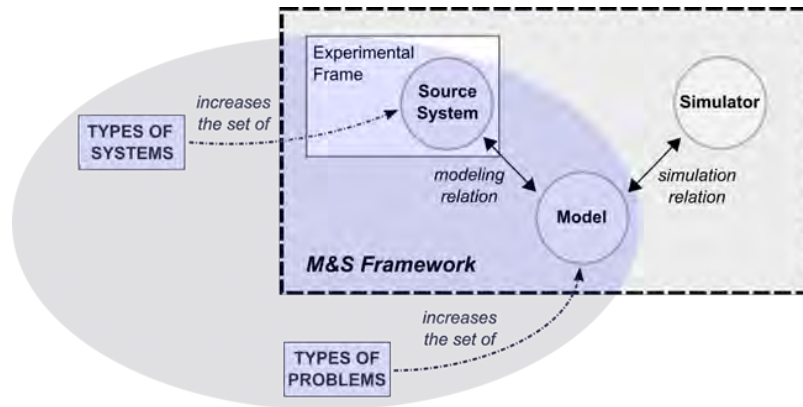

Figure 3: Relations between the M&S framework and the dimensions used to classify the DEVS extensions.

3   DEVS CONCEPTUAL FRAMEWORK

The DEVS Conceptual Framework (DEVSCF) is designed as a set of concepts, relationships and properties that provides a multilevel approach to analyze the mandatory properties required for each type of DEVS extension. The core of the metamodel is based on the two dimensions proposed in Section II. The final conceptualization of framework is presented in Figure 4 (using UML as modeling language).

As the model shows, the DEVSCF includes three abstraction layers designed as a multilevel structure from some of the concepts included in the M&S Framework: *Standard Modeling*, *Specific Modeling*, and *Modeling Specification*. The *Standard Modeling* level uses a subset of components from the M&S Framework in order to illustrate the modeling task. In the *Specific Modeling* level, the DEVSCF defines two concepts called *Dynamic System* and *Discrete Event Model* to represent *Source System* and *Model* entities, respectively. It also redefines the *modeling relation* as *DEVS modeling relation* in order to denote specifically the discrete event modeling task that determines when a *Discrete Event Model* can be said to be a valid representation of some specific *Dynamic System*. The cardinality of the relation indicates that a *Dynamic System* can be related with several instances of the *Discrete Event Model* concept but an instance of *Discrete Event Model* (that is, a specific model) must always be linked to a single instance of the *Dynamic System* (because models are designed to represent a specific system). This constraint ensures that a model will only be related with one dynamic system and, that a dynamic system will be represented by at least one model. The *Discrete Event Model* concept is related to the *Discrete Event Specification* concept by mean of the *formalize* relationship. The participation of *Discrete Event Model* in relation with the *model* role is mandatory because all instances of *Discrete Event Model* must be related with some type of *Discrete Event Specification* in order to be detailed as a formal model. However, the participation of the *Discrete Event Specification* concept in the *formalize* relationship is optional because, theoretically, a specification can be defined without building any model.

Finally, at the *Modeling Specification* level (that is, the core of the DEVSCF), the approach defines a taxonomy of discrete event specifications. The *Discrete Event Specification* concept is a generic concept used to classify the DEVS domain. The generalization set defined from *Discrete Event Specification*

includes two concepts: *DEVS* and *DEVS Extension*. The *DEVS* concept represents the original DEVS formalism while the *DEVS Extension* concept is used to abstract the set of approaches developed from DEVS. This generalization set is: i) disjoint; because non instance of any specific concept may also be an instance of the other concept (i.e. there is no overlapping between DEVS and its extensions), and ii) incomplete; because there could be types of discrete event specifications that could not be classified as any of the specific concepts identified in the generalization set. The *DEVS Extension* concept is classified as *DEVS Variant* and *DEVS Subclass* in order to represent the different types of extensions. This generalization set is defined as: i) complete; because every instance of the general concept will be also an instance of (at least) one of the specific concepts, and ii) overlapping; because the set of specific concepts could share common instances (i.e. an extension of DEVS could be considered, at the same time, as DEVS variant and DEVS subclass, structuring the extension type defined as hybrid DEVS).
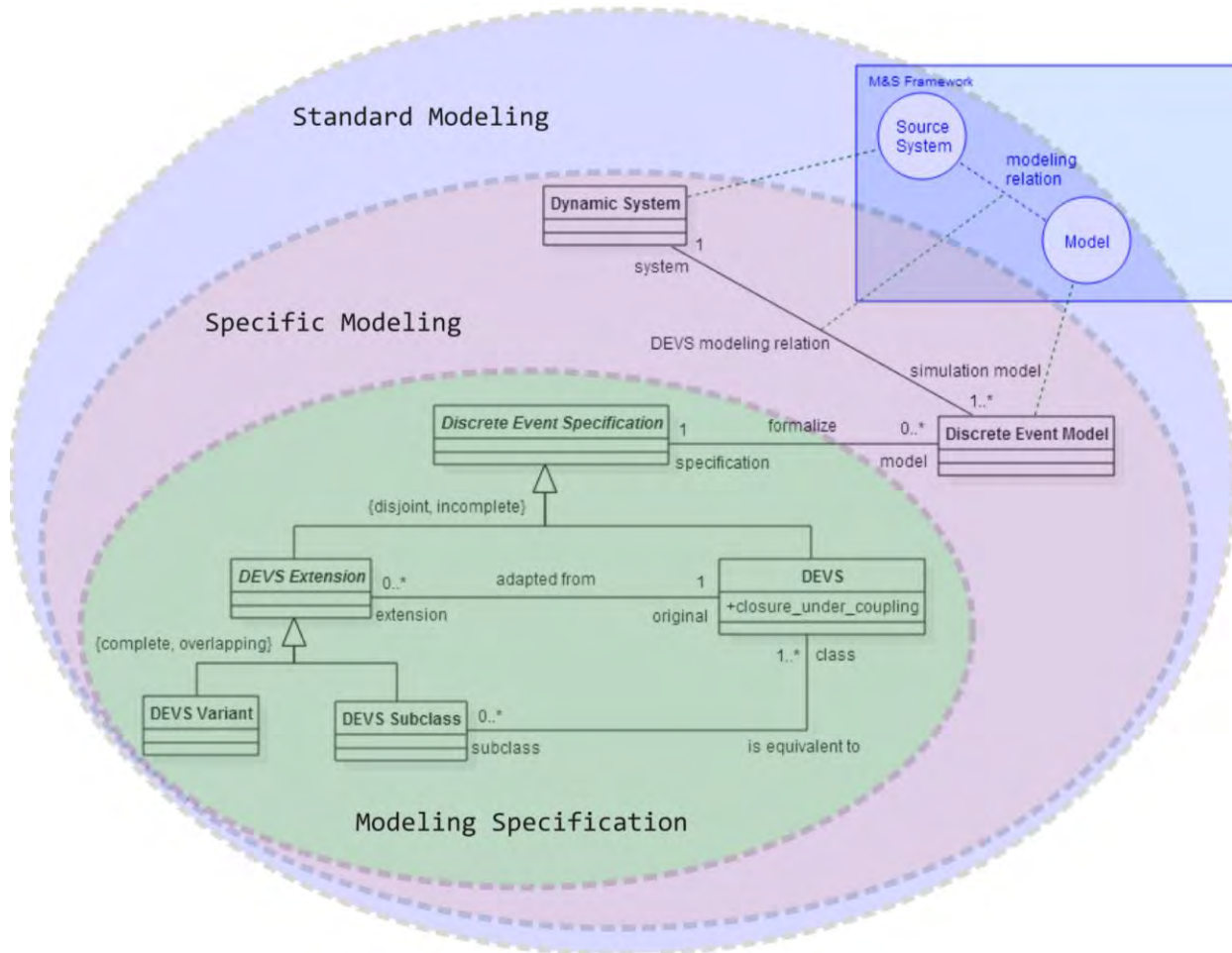

Figure 4: DEVS conceptual framework.

Besides the taxonomy modeled for the *Discrete Event Specification* concept, the *Modeling Specification* layer includes two relations designed to maintain the features required in each extension type: *adapted from* and *is equivalent to*. The *adapted from* relationship is mandatory for the *DEVS Extension* concept because the original formalism is always used as a foundation in order to get a new approach focused on DEVS. Therefore, the extensions of DEVS can be seen as adaptations of the original formalism. Meanwhile, the *is equivalent to* relationship is used to detail the "class-subclass" dependency. The *DEVS Subclass* concept refers to a specific type of *DEVS Extension* that must be interpreted as a

specialization of the original formalism. However, if this association is modeled with a subsumption relationship, the framework will suggest that all new formalism derived from DEVS that could be classified as *DEVS Subclass*, will also have to be classified as *DEVS*. Given that this approach is incorrect, the *is equivalent to* relationship is used to define that all instances of the *DEVS Subclass* concept must have, at least, one equivalent design in *DEVS*.

3.1 Using the Conceptual Framework: Features Required for each Type of DEVS Extension

The DEVSCF presents an abstraction that distinguishes the main types of DEVS extensions by using features (e.g. attributes and relationships) to link each conceptual type with the original formalism in several ways. Then, an extension of DEVS can only by classified in terms of the proposed dimensions, if and only if, it satisfies the set of properties defined for each type.

Following the conceptual framework, the main features required in each case can be summarized as properties, namely:

- Any extension of DEVS must be developed as an adaptation of the original formalism (according to the *adapted from* relationship).
- An extension of DEVS can only be formalized as a variant or subclass (by the *complete* property defined in the generalization set).
- A variant of DEVS is considered an extension of DEVS (by the subsumption relationship modeled between *DEVS Variant* and *DEVS Extension*).
- A subclass of DEVS is considered an extension of DEVS (by the subsumption relationship modeled between *DEVS Subclass* and *DEVS Extension*).
- An extension of DEVS can be categorized as variant and subclass at the same time (by the *overlapping* property defined in the generalization set).
- Any subclass of DEVS must be reducible to (at least) one equivalent model formalized in DEVS (according to the *is equivalent to* relationship).
- Any subclass of DEVS must be more specific than DEVS (because, in conceptual modeling theory, the subtyping used for represent class-subclass dependence reflects a more detailed specification level of the concepts) (Parsons and Wand 1997).

This set of properties can be used to identify each extension type. The formal definitions of the types called *DEVS variant* and *DEVS subclass* are presented in Definition 1 and Definition 2, respectively.

Definition 1 A *variant of DEVS* is an extension of DEVS that maintains the closure under coupling property in order to represent an adaptation of the original formalism useful to describe a new type of system model.

Definition 2 A *subclass of DEVS* is an extension of DEVS designed as an adaptation of the original formalism that maintains the closure under coupling property but, at the same time, is more specific than DEVS in some context and can be reduced to (at least) one equivalent DEVS model.

These definitions suggest that the closure under coupling property is used as requirement over the extension in order to ensure its adaptation to the formalism. Besides, in the case of DEVS variants, the definition requires that the adaptation deploys a new type of system model. Meanwhile, for DEVS subclasses, the definition requires two extra features: i) the existence of some equivalence in DEVS and, ii) the comparison between DEVS and the subclass in order to prove that the extension is more specific than DEVS. Both features are related. To satisfy i), the modeler must prove that each model proposed as part of the extension has an equivalent model in DEVS (i.e. an atomic or coupled model that exhibits the same behavior or structure respectively). Then, to satisfy ii), the modeler must measure several properties of both models (the subclass and the DEVS equivalent model) in order to compare both approaches. The

set of metrics to be used must be defined for each case because, it depends on the type of improvement implemented in the subclass over the original formalism. Frequently, direct metrics as number of models and number of couplings should be useful to describe the main characteristics of the models. However, other static metrics (direct or indirect) can be defined.

4   ROUTED DEVS AS A SUBCLASS OF DEVS

The Routed Discrete Event System Specification (RDEVS) formalism is an extension of DEVS designed to hide the handling of routing information among DEVS models. It defines three types of models: i) An *essential model* that provides the behavior of some component involved in the routing process. ii) A *routing model* that defines the basic simulation entity where the routing process takes place. iii) A *network model* that describes a routing process with a specific objective that requires send/receive identified events. Then, each type of model represents an abstraction level used to depict the set of elements required to define a routing process. The lower level (that is, the RDEVS essential model) was defined as a DEVS atomic model in order to maintain DEVS formalism as a foundation of RDEVS (or vice versa, to design RDEVS as an extension of DEVS). A full description of the RDEVS models along with the closure under coupling prove are detailed in (Blas et al. 2017).

From a traditional perspective, RDEVS can be considered an extension of DEVS that provides a scalable solution to arrange the events flow. However, the following sections shows how the RDEVS formalism satisfy the set of features required to be considered as a *subclass of DEVS*. Given that the features were obtained from the conceptualization depicted in the DEVSCF, the example shows how the framework can be used to study extension types in order to ensure the required properties for each case.

By analyzing the proposed features, this section shows that RDEVS formalism is an extension of DEVS that can be classified as subclass and, therefore, it has full compatibility with DEVS models and simulators. That is, given that a subclass is a specialization of its class, the RDEVS models can be considered as DEVS models that improve the solution of a simulation model for a specific situation with a simpler modeling proposal. When the events flow is independent of the components behavior, the modeler can use RDEVS formalism to design the simulation model applying an appropriated separation of concerns, given by:

- The use of essential models to represent the behavior of the components involved in the (routing) process.
- The use of routing and network models to set up the (routing) context in which the components (modeled as RDEVS essential models) are involved.

Although DEVS formalism can be used to solve this type of problem, the use of a DEVS subclass improve the final simulation models by allowing to combine DEVS models (i.e. a DEVS atomic model as RDEVS essential model) with RDEVS models. This allows to use RDEVS formalism as a "layer" above the DEVS that provides routing functionality.
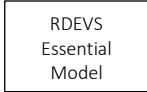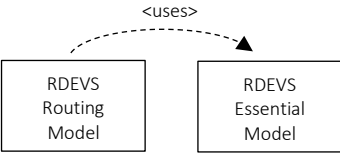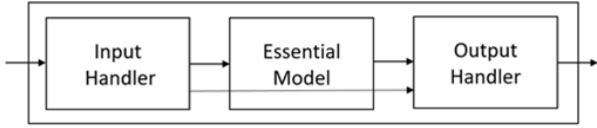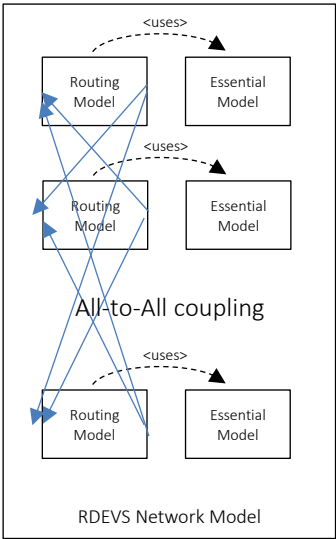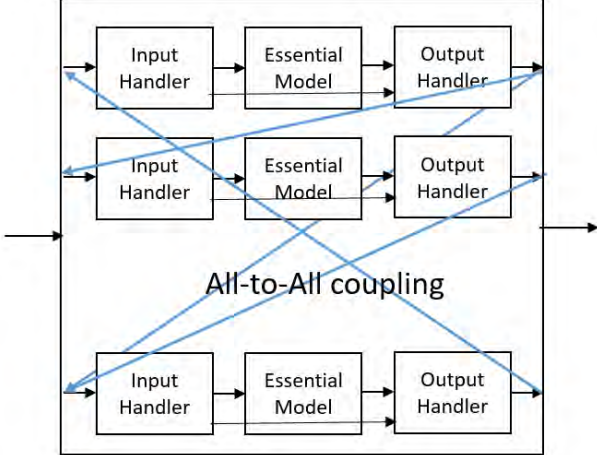
4.1   Feature #1: Closure Under Coupling

A system formalism is closed under coupling if the resultant of any network of systems specified in the formalism is itself a system in the formalism (Zeigler et al. 2000). This property ensures the hierarchical composition of the models because it allows to build models recursively with any arbitrary levels of hierarchy in a modular way.

Blas et al. (2017) prove that RDEVS formalism is closed under coupling. However, although the RDEVS network model is a well-defined DEVS coupled model, it is not necessarily equivalent to a RDEVS routing model. Then, closure under coupling shows how to define an equivalent RDEVS routing model for any RDEVS network model by building a model behaviorally equivalent to it. So, RDEVS models can be built hierarchically because: i) an equivalent routing model can be obtained from the network model and, ii) an equivalent essential model can be obtained from the routing model.

4.2 Feature #2: DEVS Equivalence

In order to ensure the DEVS equivalence, it is necessary to define a set of DEVS models that exhibit the same behavior or structure that RDEVS models (Table 1). Hence, each type of model defined as part of the RDEVS formalism needs to be represented by some DEVS model (atomic or coupled).

Table 1: DEVS equivalent models used to represent RDEVS models.

Type of Model	RDEVS Model	DEVS Equivalent Model
Essential model	RDEVS Essential Model	DEVS Atomic Model
Routing model		
Network model		

However, the set of models proposed to ensure the equivalence property should keep the dependencies designed among RDEVS models. That is, each type of RDEVS model was designed as a component that abstracts some element used in the composition of other routing process elements. From this point of view, the dependencies among RDEVS models can be summarized as:

- The *RDEVS network model* is composed by a set of *RDEVS routing models* (i.e. the definition of a network model involves detailing the set of routing models used as components of the network - composition relationship-).
- The *RDEVS routing model* embeds a *RDEVS essential model* (i.e. the definition of a routing model requires attaching an essential model used as behavioral description of the routing component -composition relationship-).

Then, the set of DEVS equivalence models must keep the composition relationships in the same way that RDEVS models.

Table 1 shows a set of DEVS models proposed by Zeigler (2018) as substitutes of RDEVS models. Given that the RDEVS essential model is defined as a DEVS atomic model, there is no need to define a new representation for this model. Moreover, both routing and network models are designed as DEVS coupled models in order to keep the composition relationships. In the routing model equivalence, the handlers take care of the routing information in the incoming message and pass on the operative content to the essential model for processing. From this perspective, the essential model is explicitly used in the routing model instead of be embedded in it.

The models proposed in Table 1 are just one possible representation of the RDEVS models using DEVS formalism. Although other representations can be designed, the set of proposed models reflects the type of components commonly used to model routing processes. The translation between RDEVS models and DEVS equivalent models is not shown in this paper for space reasons.

4.3 Feature #3: Comparison between RDEVS vs DEVS

The two approaches proposed in Table 1 are useful to model routing processes. However, if both approaches are useful, there should be some benefits of using RDEVS instead of DEVS. Several metrics can be used to analyze such benefits. However, in order to illustrate a comparison set, this paper analyze the models detailed in both formalisms applying two universal metrics: number of models and number of couplings. Table 2 shows the measures obtained for each type of model (highlighting the lower values).

Table 2: Metrics applied to RDEVS and DEVS models.

Model	RDEVS Model		DEVS Equivalent Model	
Essential model	# of models	1	# of models	1
	# of couplings	0	# of couplings	0
Routing model	# of models	2	# of models	4
	# of couplings	0	# of couplings	5
Network model *(where n is the number of routing models that compose it)*	# of models	$(2 \times n)+1$	# of models	$(4 \times n)+1$
	# of couplings	$n \times (n-1)$	# of couplings	$n \times (n-1)+(5 \times n)$

At least from the modeling perspective, the RDEVS models seems to be less complex than DEVS models. As the table shows, the RDEVS formalism improves the description of each type of model by reducing the number of models and couplings involved for each case. The impact of using RDEVS instead of DEVS in larger simulation models will be a lower number of components with less couplings among them (which gives a simpler simulation model composition). Less models and couplings implicitly reduce the processing required to execute the simulation. Then, RDEVS simplifies the simulation model required to solve a routing problem.

5 CONCLUSIONS AND FUTURE WORK

With a growing number in new extensions of DEVS and an increasing number of problems to be solved using discrete simulation techniques, the connections among different approaches has become fuzzy. Despite various existing classifications of DEVS, this paper presents a strategy that considers the context as a factor for developing a typology of DEVS formalism and its extensions.

This work discusses a conceptual framework designed to represent the domain of DEVS extensions in order to allow studying the features required for each extension type. Two types of extensions are identified: variants and subclasses. While DEVS variants refer to the extensions of DEVS formalism created to represent new types of systems, the subclasses of DEVS are alternative formalisms that improve the solution of the simulation problem applying DEVS models in a meaningful way. By using

the proposed framework, a set of features can be depicted in order to analyze both types of DEVS extensions. To illustrate the proposed guidelines, the RDEVS formalism is studied from the subclass perspective. The main achievement related with this classification is that only the models developed using a subclass of DEVS can be combined with DEVS to improve the final simulation model (because DEVS subclasses refer to DEVS specializations while variants of DEVS are designed as new formalizations). Then, this type of extensions allows executing its models using a DEVS simulator.

Future research can take advantage of the fact that a mathematical characterization of Dynamic Systems exists (Wymore 1967) and has been formalized in (Zeigler 1976). The cases where DEVS has been combined with other formalisms requires a deeper analysis prior to be included in the framework.

The DEVSCF can be used as a vehicle to study DEVS domain in order to analyze other types of properties. To get a more detailed classification, the framework can be extended by adding new dimensions or levels for each factor. Moreover, the classification scheme proposed using the problem and system dimensions defines a background in which both aspects can be measured in order to get an appropriate comparison of several extensions. That is: ¿when an extension of DEVS is highly domain dependent? ¿when it is context exclusive?. These questions can only be addressed if metrics and indicators are defined in order to evaluate the proposed dimensions over each class of extension. Then, the framework gives a structure useful to develop evaluation mechanisms for DEVS extensions.

REFERENCES

Abdelmegid, M. A., V. A. González, A. M. Naraghi, M. O'Sullivan, C. G. Walker, and M. Poshdar. 2017. "Towards a Conceptual Modeling Framework for Construction Simulation". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan et al., 2372-2383. Piscataway, New Jersey: IEEE.

Barros, F. J. 1997. "Modeling Formalisms for Dynamic Structure Systems". *ACM Transactions on Modeling and Computer Simulation* 7(4):501-515.

Blas, M. J., S. Gonnet, and H. Leone. 2017. "Routing Structure over Discrete Event System Specification: A DEVS Adaptation to Develop Smart Routing in Simulation Models". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan et al., 774-785. Piscataway, New Jersey: IEEE.

Furian, N., M. O'Sullivan, C. Walker, S. Vössner, and D. Neubacher. 2015. "A Conceptual Modeling Framework for Discrete Event Simulation using Hierarchical Control Structures". *Simulation Modelling Practice and Theory*, 56(1):82-96.

Guizzardi, G., and T. Halpin. 2008. "Ontological Foundations for Conceptual Modelling". *Applied Ontology*, 3(1):1-12.

Hong, J., H. Song, T. Kim, and K. Park. 1997. "A Real-Time Discrete Event System Specification Formalism for Seamless Real-Time Software Development". *Discrete Event Dynamic Systems* 7(4):355-375.

Kwon, Y., H. Park, S. Jung, and T. Kim. 1996. "Fuzzy-DEVS Formalism: Concepts, Realization and Applications". *International Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, March 23-37 1996, San Diego, CA, USA, 227-234.

Olivé, A. 2007. *Conceptual Modeling of Information Systems*. Heidelberg: Springer-Verlag Berlin Heidelberg.

Parsons, J., and Y. Wand. 1997. "Choosing Classes in Conceptual Modeling". *Communications of the ACM* 40(6):63-69.

Robinson, S. 2006. "Conceptual Modeling for Simulation: Issues and Research Requirements". In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone et al., 792-800. Piscataway, New Jersey: IEEE.

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

Robinson, S., G. Arbez, L. G. Birta, A. Tolk, and G. Wagner. 2015. "Conceptual Modeling: Definition, Purpose and Benefits". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz et al., 2812-2826. Piscataway, New Jersey: IEEE.

Wainer, G. A., and P. J. Mosterman. 2010. *Discrete-Event Modeling and Simulation: Theory and Applications*. Florida: CRC Press Inc.

Wymore, A. W. 1967. *A Mathematical Theory of Systems Engineering: The Elements*. New York: John Wiley & Sons.

Yilmaz, L., and T. I. Ören. 2004. "A Conceptual Model for Reusable Simulations within a Model-Simulator-Context Framework". *Conference on Conceptual Modelling and Simulation*, October 28-31 2004, Genoa, Italy, 235-241.

Zeigler, B. P. 1976. *Theory of Modeling and Simulation*. 1st ed. New York: Wiley Interscience.

Zeigler, B. P. 2018. "Closure Under Coupling: Concept, Proofs, DEVS Recent Examples". *Spring Simulation Multi-conference - Theory of Modeling and Simulation Symposium*, April 15-18 2018, Baltimore, MD, USA.

Zeigler, B. P., and S. Chi. 1992. "Symbolic Discrete Event System Specification". *IEEE Transactions on Systems, Man, and Cybernetics* 22(6):1428–1443.

Zeigler, B. P., and S. Vahie. 1993. "DEVS Formalism and Methodology: Unity of Conception/Diversity of Application". In *Proceedings of the 1993 Winter Simulation Conference*, edited by G. W. Evans et al., 573–579. Piscataway, New Jersey: IEEE.

Zeigler, B. P., and H. S. Sarjoughian. 2003. "Introduction to DEVS Modeling and Simulation with Java: Developing Component-based Simulation Models". Technical Document, Arizona Center for Integrative Modeling and Simulation, University of Arizona.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. 2nd ed. London: Academic Press.

AUTHOR BIOGRAPHIES

MARÍA J. BLAS received her Information Systems Engineering degree from Universidad Tecnológica Nacional (UTN) in 2014. She has a PhD Research Fellowship from Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) and works at Instituto de Desarrollo y Diseño INGAR. Her thesis is focused on discrete-event simulation in software architectures. Her e-mail address is mariajuliablas@santafe-conicet.gov.ar.

SILVIO M. GONNET received his PhD degree in Engineering from UNL in 2003. He currently holds a Researcher position at CONICET and works at INGAR. Also, he works as an Assistant Professor at UTN. His research interests are models to support design processes and conceptual modeling. His e-mail address is sgonnet@santafe-conicet.gov.ar.

HORACIO P. LEONE is a Full Professor at UTN. He also holds a Researcher position at the CONICET, working at INGAR. He obtained his PhD degree in Chemical Engineering from UNL in 1986 and was a Postdoctoral Fellow at the Massachusetts Institute of Technology. His current research activities focus on software architectures, models for supporting the design process, and enterprise modelling. His email address is hleone@santafe-conicet.gov.ar.

BERNARD P. ZEIGLER is Chief Scientist at RTSync Corp., Professor Emeritus of Electrical and Computer Engineering at the University of Arizona, Tucson and co-Director of the Arizona Center for Integrative Modeling and Simulation. He is internationally known for his contributions in modeling and simulation theory. He develop the Discrete Event System Specification (DEVS) formalism in 1976. He has published several books including "Theory of Modeling and Simulation" and "Guide to Modeling and Simulation of Systems of Systems". His email address is zeigler@rtsync.com.