Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

ROUTING STRUCTURE OVER DISCRETE EVENT SYSTEM SPECIFICATION: A DEVS ADAPTATION TO DEVELOP SMART ROUTING IN SIMULATION MODELS

María Julia Blas
Silvio Gonnet
Horacio Leone

Instituto de Diseño y Desarrollo INGAR
Universidad Tecnológica Nacional – Consejo Nacional de Investigaciones Científicas y Técnicas
Avellaneda 3657
Santa Fe, 3000, ARGENTINA

ABSTRACT

The Discrete Event System Specification (DEVS) formalism has become an engine for advances in modeling and simulation technology. Many extensions of DEVS formalism have been developed across the years in order to solve different types of situations. However, when the acceptance of input events and the generation of output events are related with model capabilities, solutions developed with current formalisms are too complex. This paper presents a new simulation formalism called Routed DEVS (RDEVS) in which routing information is used to manage directed events. The behavior supported by the new formalism is useful to create simulation models of web application architectures. However, it could also be applied to other contexts. The RDEVS formalism is based on DEVS and is closure under coupling (i.e. models can be built hierarchically). The formal specification of RDEVS formalism and a briefly description of its framework implementation are presented in this work.

1 INTRODUCTION

The Discrete Event System Specification (DEVS) is a modeling formalism based on systems theory that provides a general methodology for hierarchical construction of reusable models in a modular way. Since its introduction in late 70s (Zeigler 1976), the formalism has been used to simulate several systems related with multiple domains, including mobile applications (Kim et al. 2016), social networks (Bouanan 2015), supply chain management (Godding and Sarjoughian 2003; Gholami et al. 2014), and software engineering (Bogado, Gonnet, and Leone 2014; Risco-Martín et al. 2016; Blas, Gonnet, and Leone 2017).

The core of DEVS includes a modeling and simulation (M&S) framework structured in three main components: model, simulator and experimental frame (Zeigler, Praehofer, and Kim 2000). The model describes the system specification, structure, and behavior. The simulator refers to the computational system that executes the instructions detailed in the model. The experimental frame represents the conditions under which the system is observed, building the experimentation and validation context to be used on the model. Each component is specified as an individual element in order to maintain its independence. Keeping independency between components provides multiple benefits to the framework, such as: the same model can be executed by different simulators, or several experiments can be changed to study different situations (Bogado, Gonnet, and Leone 2014). Given that components needs to interact with each other's in order to accomplish its functions, the framework includes a set of relationships that allows defining these interactions. Some of the relations included involve modeling relation and simulation relation. While 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 (Wainer and Mosterman 2010).

Over the years, DEVS has finding an increasing acceptance in the model-based simulation research community becoming one of the preferred paradigms to conduct modeling and simulation enquiries (Wainer and Mosterman 2010). Following the approach proposed in the M&S framework, new variants, extensions and abstractions have been developed using the core of concepts defined by the original formalism. Several authors have improve the formalism capabilities in response to different situations, giving solutions useful to a wide range of simulation problems. Some of these solutions include Cell-DEVS (Wainer 2004), Dynamic Structure DEVS (Barros 1995), Fuzzy-DEVS (Kwon et al. 1996), Min-Max-DEVS (Hamri, Giambiasi, and Frydman 2006), Parallel DEVS (Chow and Zeigler 1994), and Vectorial DEVS (Bergero and Kofman 2014). Moreover, the separation of concerns in the model/simulator components included in the M&S framework allowed researchers to develop alternative simulations algorithms in order to complement the existent solutions (Kim, Kim, and Park 1998; Muzy and Nutaro 2005; Shang and Wainer 2006; Liu 2010; Liu and Wainer 2010). So, it is evident that DEVS formalism grows along with the evolution of the problems treated with discrete event simulation techniques. As problems complexity increase, new mechanisms are required to deal with them.

Cloud computing (CC) has recently emerged as a new paradigm for hosting and delivering services over the internet (Zhang, Cheng, and Boutaba 2010). Conceived as a new discipline of software engineering, CC presents a new set of problems that must be solve using new or improved techniques. That is the case of web applications (WA). Actually, the main research topics related to WA involve different areas, such as: requirements elicitation (Valderas and Pelechano 2011, Breaux and Rao 2013), architectural evaluation (Kossmann, Kraska, and Loesing 2010; Wallis, Henskens, and Hannaford 2010) and testing (Artzi et al. 2010; Nguyen, Kästner, and Nguyen 2014; Li, Andreasen, and Ghosh 2014). In this context, this paper proposes a DEVS adaptation called Routed DEVS (RDEVS) designed to support the simulation of WA components in order to study its behaviors. This new formalism defines a set of models that use routing information to identify the events source and destinations. According to this information, models determines the acceptance of input events before execute its instructions. Also, models are capable to send output events to a specific group of receptors or the same event to different receptors according to its actual state. Although the proposal is centered on WA, the RDEVS formalism can be also applied to similar problems from other contexts (e.g. mobile applications and systems-of-systems).

The remainder of this paper is organized as follows. Section II describes the CC environment, highlighting the WA context and the problems detected when WA community requires discrete event simulation models. Section III presents the RDEVS formalism including the models formal specifications. Section IV proves the closure under coupling of the RDEVS formalism in order to guarantee the hierarchical composition of the models. Section V introduce the software framework developed to support the RDEVS formalism. Finally, Section VI is devoted to conclusions and future work.

2 RELATED WORK

CC is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance 2011). Usually, CC architectures are designed using the layered architecture pattern. Blas et al. (2016a) have proposed a five layers architecture (adapted from literature) in which end users interact with WA located at the application layer. This layer is at the top of the proposed architecture and allows consumers access to applications installed on the data-centers of a cloud provider. Then, WA architectures should be deployed on this layer.

The software architecture design can be considered the earliest design specification of any software product. Since this design is composed of a set of components and its connections, it can be used as a vehicle to predict and estimate the final behavior of a software product (Kruchten, Lago, and Van Vliet 2006; Giesecke, Hasselbring, and Riebisch 2007; Koziolek et al. 2008; Roldán, Gonnet, and Leone 2013;

Li, Liang, and Avgeriou 201; Bogado, Gonnet, and Leone 2014). Applying this approach at CC level, an integrated approach to analyze the behavior of WA can be built by combining architectural designs and simulation techniques. If the set of available elements to specify a WA architecture is defined (e.g. using a metamodel), a simulation model can be build applying a systematic transformation from each architectural component into a simulation model. Then, designers could evaluate different WA architectures over the same infrastructure before to move to the next step of the development process. Also, they could analyze the impact of architectural changes over the final product quality and make recommendations based on this evaluation. Given that WA systems are discrete event systems, DEVS formalism can be used to define such simulation models. Then, WA architectures can be study using discrete event simulation models. A full description of this approach was presented in a previous work (Blas, Gonnet, and Leone 2016b).

A quick analysis of the DEVS simulation models proposed in (Blas, Gonnet, and Leone 2016b), reflects immediately the complexity of their structure. Bass et al. (2012) defines software architecture as the structure, or structures, of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. According to this definition, software architectures illustrates a set of components linked by connections. These connections are based on the software elements relationships required to solve all possible requests. Then, connections are derived from the components interaction that have place when software needs to solve an specific request. So, the connections specification is defined at architectural level but its definition is attached to the request types involved in the software. When software architecture is used as unique source of the software specification to build a simulation model, the final simulation model reflects all connections without knowing which components are required to solve a specific request type. Then, the simulation models instantiated during the transformation process must manage the complexity of the requests flows. This is the approach used by Blas et. al (2016b). However, although this is a good approach, is not scalable. When the number of architectural components grow, the configuration of the simulation models involved in the transformation becomes a problem since the designer must know the flows of the requests. Clearly a new mechanism, that allows to manage this type of behavior inside the models without increasing the complexity of the structure, is required.

To this purpose, the following sections introduce an adaptation of DEVS formalism called Routed DEVS (RDEVS) in which models are capable to manage the flow of events using their state information. Following the RDEVS formalism, simulation models can be designed to reflect only the desired behavior of the component without define explicitly all possible connections between them. In the WA context, if input events represent different end-user requests that navigate inside the architectural components, the RDEVS architectural model will manage the flow of each request type using the set of connections specified at architectural level. RDEVS internal models will reflect architectural components that can decide to accept/deny requests using routing information. Then, by using RDEVS formalism, simulation models will have the knowledge to determinate the treatment of the requests.

3 ROUTED DEVS FORMALISM

The Routed Discrete Event System Specification (RDEVS) formalism is an extension of the DEVS formalism based on the premise that a simulation model must process only the set of events that comes from authorized senders and have been sent specifically to it. To guarantee this premise, the formalism requires that each simulation model have an unique identifier that defines its existence as part of the routing process. A simulation model will accept input events if and only if: i) the sender identifier is an authorized model; ii) the model identifier is included in the set of possible receptors of the event. Then, all events of RDEVS include the information required to identify its sender and all possible receptors.

The RDEVS formalism defines three types of models: essential model, routing model and network model. Each model represent an abstraction level used to define different elements required as part of the routing process.

3.1 RDEVS Essential Model

Essential model specifies the behavioral description of a simulation model that represents a component involved in the routing process. Formally, essential models are defined in the RDEVS formalism, as an atomic model of the DEVS formalism, by the structure

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau >$$

where

$X \equiv$ set of input events,
$S \equiv$ set of sequential states,
$Y \equiv$ set of output events,
$\delta_{int}: S \rightarrow S \equiv$ internal transition function,
$\delta_{ext}: X \times Q \rightarrow S \equiv$ 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,
$\lambda: S \rightarrow Y \equiv$ output function,
$\tau: S \rightarrow R_o^+ \equiv$ time advance function.

3.2 RDEVS Routing Model

Routing model defines the basic simulation model where the routing process has place. In order to accept/deny input events and redirect output events, the model includes a set of elements that depicts the routing information. It uses the behavioral specification of an essential model in order to define its own specification over the routing process. That is, the routing model encapsulates the description of an essential model with the routing specification required to determine the events occurrence. Multiple routing models can use the same essential model with different routing information in order to create different events routing processes.

Formally, routing models are defined by the structure

$$R = < \omega, E, M >$$

where

$\omega = (u, W, \delta_r) \equiv$ routing information, where
  $u \in N_0 \equiv$ model identifier,
  $W = \{ w_1, w_2, \ldots, w_p \mid w_1, w_2, \ldots, w_p \in N_0 \} \equiv$ set of source model identifiers that represents the allowed routing models of R (from which can receive events),
  $\delta_r: S_M \rightarrow T \equiv$ routing function used to direct output events, where S_M is defined on M and
    $T = \{ t_1, t_2, \ldots, t_k \mid t_1, t_2, \ldots, t_k \in N_0 \} \equiv$ set of destination model identifiers,
$E = < X_E, S_E, Y_E, \delta_{int,E}, \delta_{ext,E}, \lambda_E, \tau_E > \equiv$ essential model used by R,
$M = <X_M, S_M, Y_M, \delta_{int,M}, \delta_{ext,M}, \lambda_M, \tau_M> \equiv$ DEVS atomic model that specifies the routing process, where
  $X_M = \{ (x, h, T) \mid x \in X_E, h \in N_0, T = \{ t_1, t_2, \ldots, t_k \mid t_1, t_2, \ldots, t_k \in N_0 \} \} \equiv$ set of identified input events, with
    $x \equiv$ input event defined in E,
    $h \equiv$ sender model identifier,
    $T \equiv$ set of target model identifiers,
  $S_M = S_E \equiv$ set of sequential states,
  $Y_M = \{ (y, h, T) \mid y \in Y_E, h \in N_0, T = \{ t_1, t_2, \ldots, t_k \mid t_1, t_2, \ldots, t_k \in N_0 \} \} \equiv$ set of identified output events, with
    $y \equiv$ output event generated by E,

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

$T \equiv$ set of target model identifiers,

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

$\delta_{ext,M}: Q_M \times X_M \rightarrow S_M \equiv$ external transition function that only accepts events that have been sent to R from some allowed model or from an external source (when u=0 and W=ø), defined as

$$\delta_{ext,M}(s,e,x') = \begin{cases} \delta_{ext,E}(s,e,x) & \text{if } (u \in T \vee u = 0) \wedge (h \in W \vee W = ø) \text{ with } x'=(x,h,T) \\ s & \text{otherwise} \end{cases}$$

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,

$\lambda_M: S_M \rightarrow Y_M \equiv$ output function that generates identified events, defined as

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

$\tau_M: S_M \rightarrow R_o^+ \equiv$ time advance function.

3.3 RDEVS Network Model

Network model describes a complex simulation model with an specific objective that requires send/receive identified events over a routing process. Its definition includes a set of routing models and, indirectly, the couplings between them in order to identify its influences. It also requires two translation functions in order to match the events from other models with the identified events used in the model. Then, a RDEVS network model can interact with (one or more) RDEVS network models or, simply, with DEVS models, according the purpose of the simulation.

Formally, network models are defined by the structure

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

where

$X \equiv$ set of input events,

$Y \equiv$ set of output events,

$D \equiv$ set of model identifiers (routing model references), where $d \in N_0, \forall d \in D$,

For each $d \in D$, R_d is a routing model, defined as

$$R_d=< \omega_d, E_d, M_d >$$

where $u_d = d$,

For each $d \in D \cup \{ N \}$, I_d is the set of influences over d where $d \notin I_d$,

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

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

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

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

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

$x \equiv$ input event defined in N,

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

$T \equiv$ set of target model identifiers that must process the input event,

T_{out}: $\{ (y, h, T) \mid y \in Y, h \in N_0, T = \{ (t_1, t_2, \ldots, t_k) \mid t_1, t_2, \ldots, t_k \in N_0 \} \} \rightarrow Y$ ≡ output translation function that takes an identified output event and returns an output event, where

y ≡ output event allowed in N,

h ≡ sender model identifier,

$T = ø$ ≡ set of target model identifiers where an empty set indicates that the event is sent to an external destination,

Select: $2^D \rightarrow$ D ≡ function for tie-breaking between simultaneous events.

4 ROUTED DEVS CLOSURE UNDER COUPLING

According to Zeigler (2000), 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. The proof of this property is a very important key in a system formalism because it ensures the hierarchical composition of the models. That is, it allows to build models recursively with any arbitrary levels of hierarchy in a modular manner.

To prove that RDEVS formalism is closed under coupling and, therefore, models can be built in hierarchical manner, is necessary to obtain:

- The equivalent RDEVS routing model from the RDEVS network model (Proof #1).
- The equivalent RDEVS essential model from the RDEVS routing model (Proof #2).

If both models can be obtained, the hierarchical composition will be allowed because a network model (treated as its behaviorally equivalent to a routing model) will be able to become component of larger network models. The same will happened with the routing model and its essential model, since the routing model (treated as its behaviorally equivalent to the essential model) will be able to be use as part of larger routing models. Then, by transitivity, a dynamic system specified by a network model will be behaviorally equivalent to an essential model and will be able to be use as part of larger routing models.

Sections 4.1 and 4.2 detail the equivalences required to ensure RDEVS closure under coupling.

4.1 Proof #1: RDEVS Network Model to RDEVS Routing Model

The network model described by the structure

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

where each $d \in D$ references a routing model R_d defined as

$$R_d = < \omega_d, E_d, M_d >$$

defines an equivalent routing model structured as

$$R = < \omega, E, M >$$

in which:

- $\omega = (u, W, \delta_r) = (0, ø, \delta_r) \mid \delta_r : S_M \rightarrow T \wedge T = ø$ ≡ routing information that represents R as equivalent model of N. Zero is used to identify the new model. Since N uses the translation functions to determine events sources and destinations, W and T are specified as an empty set.
- $E = < X_E, S_E, Y_E, \delta_{int,E}, \delta_{ext,E}, \lambda_E, \tau_E >$ ≡ essential model used by R in which:
    - $X_E = X$ ≡ set of input events of E defined as the set of input events of N. Given that R and N are equivalent and R is composed by E, the inputs must be the same.

- $S_E = \times_{i \in D} Q_i \mid Q_i = \{ (s_i, e_i) \mid s_i \in S_{M,i},\ 0 \le e_i \le \tau_{M,i}(s_i),\ \forall i \in D$ ≡ set of sequential states of E specified as the product of the Q_i sets defined for each model that compose N. Each Q_i is defined as an ordered pair that contains the state and the elapsed time of the R_i model.
- $Y_E = Y$ ≡ set of output events of E defined as the set of output events of N. As in the input events, the outputs must be the same.
- $\delta_{int,E}(s) = s'$ ≡ internal transition function of E that transforms different parts of the state, where $s = (\ldots,(s_j, e_j),\ldots)$ and $s' = (\ldots,(s'_j, e'_j),\ldots)$. Given that the state of E is defined using the set of models included in N, an internal transition of E may involve simultaneous internal transitions of multiple components. Then, considering that the imminent components are collected in a set structured as

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

one model i^* must be selected in order to execute its internal transition. This selection can be done using the tie-breaking function of N. Then $i^* = Select(IMM(s))$ and, therefore, the imminent internal transition to be executed belongs to the R_{i*} model. However, the execution this transition will bring the execution of all external transitions of the components influenced by R_{i*}. So, the state transformation is specified as

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

while the elapsed time transformation is defined as

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

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

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

so the state transformation is defined as

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

- $\lambda_E(s)\colon S_E \rightarrow Y_E$ ≡ 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 the outputs of N, specified as

$$\lambda_E(s) = \begin{cases} Z_{i*,N}\left(\lambda_{M,i*}(s_{i*})\right) & si\ N \in I_{i*} \\ \emptyset & otherwise. \end{cases}$$

- $\tau_E: S_E \rightarrow R_o^+$ ≡ time advance function of E that select the most imminent event time of all the components included in N. This function is defined as $\tau_E(s) = min \{ \sigma_i \mid i \in D \}$ where $\sigma_i = \sigma_{M,i}(s_i) - e_i$.

- $M = < X_{M,} S_M, Y_M, \delta_{int,M}, \delta_{ext,M}, \lambda_M, \tau_M >$ ≡ DEVS atomic model that specifies the routing process of R. M definition is depicted at routing model level, specifying the routing process using components of E as M elements. Then, the equivalence transformation of N into R, it cannot change this specification. M must follow its definition using the components of E defined above.

4.2 Proof #2: RDEVS Routing Model to RDEVS Essential Model

The routing model described by the structure

$$R = < \omega_R, E_R, M_R >$$

with $M_R = < X_{M,R,} S_{M,R}, Y_{M,R}, \delta_{int,M,R}, \delta_{ext,M,R}, \lambda_{M,R}, \tau_{M,R} >$, defines an equivalent essential model structured as

$$M = <X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau>$$

in which $X = X_M$, $S = S_M$, $Y = Y_M$, $\delta_{int} = \delta_{int,M}$, $\delta_{ext} = \delta_{ext.M}$, $\lambda = \lambda_M$ and $\tau = \tau_M$.

According to this equivalence proof, each component of the M_R model that composes a RDEVS routing model can be directly mapped to a new model component that defines a RDEVS essential model. This equivalence is valid because both models (M and M_R) are based on the structure of a DEVS atomic model. Since the specification of M_R uses the content of ω_R y E_R, all the information required to execute the routing process will be also included in M.

5 ROUTED DEVS SOFTWARE FRAMEWORK

A software framework implementation of the RDEVS formalism was developed in order to provide a modeling and simulation environment to fully support the implementation of routing structures over discrete event systems. Given that RDEVS is based on DEVS formalism, RDEVS models can be executed using DEVS simulation algorithm and experimental frame. Then, RDEVS implementation can extend existent implementations of DEVS formalism.

In this context, the proposal was to develop the RDEVS software framework (RDEVS_SF) as an extension of DEVSJAVA (Arizona Center for Integrative Modeling and Simulation 2005). The set of Java classes implemented involve several concepts related to RDEVS formalism, such as:

- *EssentialModel.java*, *NetworkModel.java* and *RoutingModel.java* to define the types of models detailed as part of RDEVS.
- *RoutingFunction.java* and *RoutingFunctionElement.java* to define the δ_r required as part of the routing information in RDEVS routing model.
- *IdentifiedEvent.java* to define the structure of all the events manipulated by the simulator.
- *TranslationFunction.java*, *InputTranslationFunction.java* and *OutputTranslationFunction.java* to define the processes required in RDEVS network model to adjust external events.

In order to use the DEVS simulation algorithm and the graphical tools available, some of the classes defined in RDEVS_SF were hierarchically related to DEVSJAVA classes. This dependency allows to use DEVS Suite (Arizona Center for Integrative Modeling and Simulation 2009) as graphical environment for RDEVS models defined in RDEVS_SF. Figure 1 shows a screenshot of a basic example implemented in RDEVS_SF where: i) project explorer shows the set of classes implemented as part of the framework; ii) DEVS Suite is used to depict models structures.

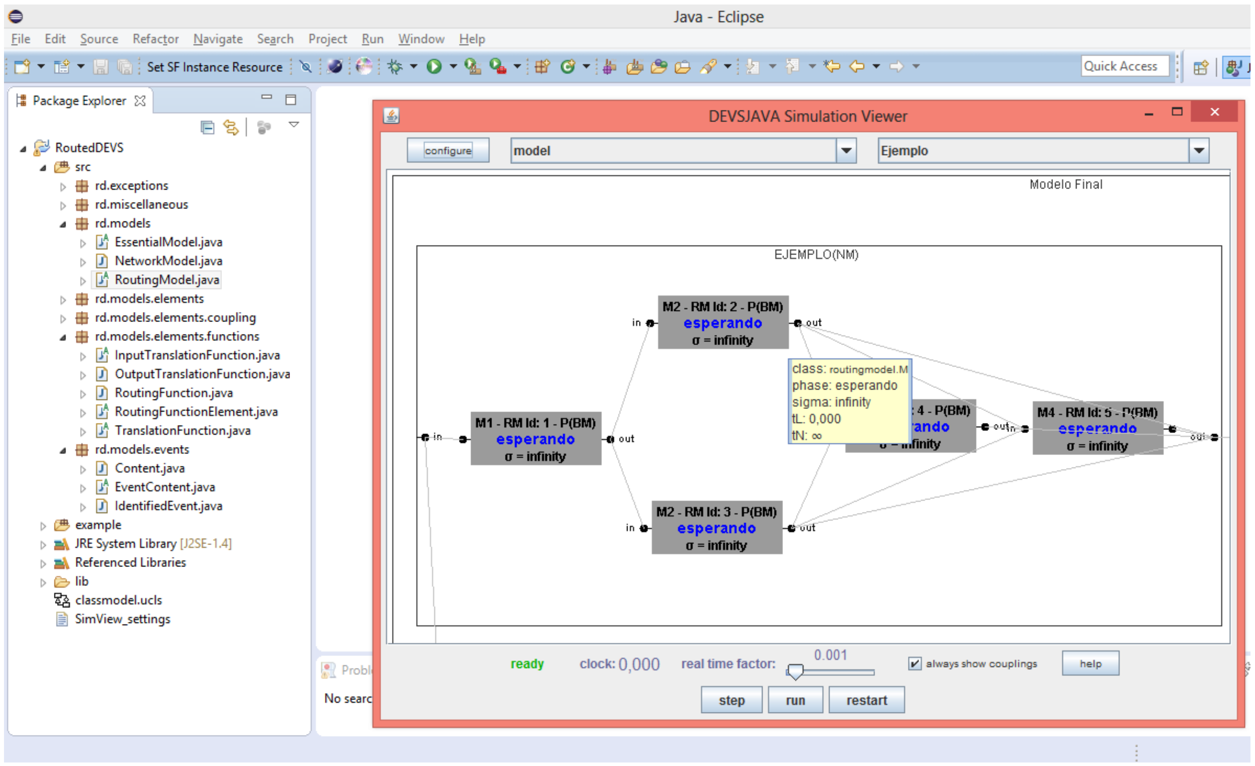

Figure 1. Simulation model example implemented in RDEVS_SF.

6 CONCLUSIONS AND FUTURE WORK

The DEVS formalism gives several benefits when is used to develop simulation models related to software engineering. However, when architectural designs must be mapped into simulation models, the complexity of manage the events flow becomes a problem. In this paper, an adaptation of DEVS was described in order to provide a scalable solution to this problem. The RDEVS formalization and its closure under coupling are the main properties of the formalism presented in this work. Also, a brief description of the software framework implemented to support the formalism proposed has been included.

Although RDEVS was developed to solve a specific modeling problem detected when web applications architectures are mapped to simulation models, it can be applied to other contexts. Similar contexts include mobile applications and hardware architectures. Another possible areas involve communication protocols and systems-of-systems.

Since RDEVS essential models are equivalent to DEVS atomic models, considering the closure under coupling of RDEVS formalism, is possible to combine RDEVS models with DEVS models. Then, more powerful simulation models can be built by combining both formalisms. Even more, existent DEVS models can be used as part of RDEVS models. The same approach can be applied to other DEVS extensions that can be reduced to DEVS atomic models.

Given the independence of components identified as part of the M&S framework proposed in DEVS, the RDEVS models can be executed using DEVS simulation algorithm. However, the design of an adapted algorithm that take advantage of the routing properties is proposed as future work.

ACKNOWLEDGMENTS

The authors wish to acknowledge the financial support received from CONICET and Universidad Tecnológica Nacional – Facultad Regional Santa Fe.

REFERENCES

Arizona Center for Integrative Modeling and Simulation. 2005. DEVSJAVA. http://acims.asu.edu/software/devsjava/.

Arizona Center for Integrative Modeling and Simulation. 2009. DEVS-Suite. http://acims.asu.edu/software/devs-suite/.

Artzi, S., A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst. 2010. “Finding Bugs in Web Applications using Dynamic Test Generation and Explicit-State Model Checking”. *IEEE Transactions on Software Engineering* 36(4):474-494.

Barros, F. 1995. “Dynamic Structure Discrete Event System Specification: A New Formalism for Dynamic Structure Modeling and Simulation”. In *Proceedings of the 1995 Winter Simulation Conference*, edited by C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 781-785. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bass L., P. Clements, and R. Kazman. 2012. *Software Architecture in Practice*. Addison-Wesley Professional.

Bergero, F., and E. Kofman. 2014. “A Vectorial DEVS Extension for Large Scale System Modeling and Parallel Simulation”. *Simulation* 90(5):522-546.

Blas, M. J., S. Gonnet, and H. Leone. 2016a. “Especificación de la Calidad en Software-as-a-Service: Definición de un Esquema de Calidad basado en el Estándar ISO/IEC 25010”. In *Proceeding of the 2016 Argentine Symposium on Software Engineering* included in the *Argentine Conference on Informatics* 135-146.

Blas, M. J., S. Gonnet, and H. Leone. 2016b. “Building Simulation Models to Evaluate Web Application Architectures”. In *Proceedings of the 2016 Latin American Symposium of Software Engineering* included in the *Latin American Computing Conference* 647-657.

Blas, M. J., S. Gonnet, and H. Leone. 2017. “Modeling User Temporal Behaviors Using Hybrid Simulation Models”. *IEEE Latin America Transactions* 15(2):341-348.

Bogado, V., S. Gonnet, and H. Leone. 2014. “Modeling and Simulation of Software Architecture in Discrete Event System Specification for Quality Evaluation”. *Simulation* 90(3):290-319.

Bouanan, Y. M. Forestier, J. Ribault, G. Zacharewicz, B. Vallespir, and N. Moalla. 2015. “Simulating Information Diffusion in a Multidimensional Social Network using the DEVS Formalism”. In *Proceedings of the 2015 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium* 63-68.

Breaux, T., and A. Rao. 2013. “Formal Analysis of Privacy Requirements Specifications for Multi-tier Applications”. In *Proceeding of the 2013 Requirements Engineering Conference* 14-23.

Chow, A. C., and B. P. Zeigler. 1994. “Parallel DEVS: a Parallel, Hierarchical, Modular Modeling Formalism”. In *Proceedings of the 1994 Winter Simulation Conference*, edited by J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 716–722. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gholami, S., H. S. Sarjoughian, G. W. Godding, D. R. Peters, and V. Chang. 2014. “Developing Composed Simulation and Optimization Models using Actual Supply-Demand Network Datasets”. In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 2510-2521. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Giesecke S., W. Hasselbring, and M. Riebisch. 2007. “Classifying Architectural Constraints as a basis for Software Quality Assessment”. *Advanced Engineering Informatics* 21(2):169-179.

Godding, G., H. Sarjoughian, and K. Kempf. 2003. “Semiconductor Supply Network Simulation”. In *Proceedings of the 2003 Winter Simulation Conference*, edited by S. E. Chick, P. J. Sanchez, D. M. Ferrin, and D. J. Morrice, 1593-1601. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hamri, M., N. Giambiasi, and C. Frydman. 2006. “Min–Max-DEVS Modeling and Simulation”. *Simulation Modelling Practice and Theory* 14(7): 909-929.

Kim, K. H., T. G. Kim, and K. H. Park. 1998. “Hierarchical Partitioning Algorithm for Optimistic Distributed Simulation of DEVS Models”. *Journal of Systems Architecture* 44(6):433-455.

Kim, Y. J., J. Y. Yang, Y. M. Kim, J. Lee, and C. Choi. 2016. “Modeling Behavior of Mobile Application Using Discrete Event System Formalism”. In *Proceeding of the 2016 Asian Simulation Conference* 40-48.

Kossmann, D., T. Kraska, and S. Loesing. 2010. “An Evaluation of Alternative Architectures for Transaction Processing in the Cloud”. In *Proceedings of the 2010 International Conference on Management of Data* 579-590.

Koziolek H., S. Becker, J. Happe, and R. Reussner. 2008. “Evaluating Performance of Software Architecture Models with the Palladio Component Model”. In *Model Driven Software Development: Integrating Quality Assurance, IDEA Group Inc.* 95–118.

Kruchten P., P. Lago, and H. Van Vliet. 2006. “Building Up and Reasoning About Architectural Knowledge”. In *Proceedings of the 2006 International Conference on Quality of Software Architectures* 43–58.

Kwon, Y., H. Park, S. Jung, and T. Kim. 1996. “Fuzzy-DEVS Formalism: Concepts, Realization and Application”. In *Proceedings of the 1996 Conference on Artificial Intelligence, Simulation and Planning In High Autonomy Systems* 227– 234.

Li, G., E. Andreasen, and I. Ghosh. 2014. “SymJS: Automatic Symbolic Testing of JavaScript Web Applications”. In *Proceedings of the 2014 International Symposium on Foundations of Software Engineering* 449-459.

Li Z., P. Liang, and P. Avgeriou. 2013. “Application of Knowledge-based Approaches in Software Architecture: A Systematic Mapping Study”. *Information and Software Technology* 55(5):777–794.

Liu Q. 2010. “Algorithms for Parallel Simulation of Large-Scale DEVS and Cell-DEVS Models”. Ph.D. thesis, Systems and Computer Engineering Department, Carleton University, Ottawa. http://cell-devs.sce.carleton.ca/publications/2010/Liu10/Algorithms.pdf.

Liu, Q., and G. Wainer. 2010. “Accelerating Large-Scale DEVS-based Simulation on the Cell Processor”. In *Proceedings of the 2010 Symposium on Theory of Modeling and Simulation: DEVS Integrative M&S Symposium* 191-198.

Mell, P. and T. Grance. 2011. The NIST definition of cloud computing. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

Muzy, A., and J. Nutaro. 2005. “Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators”. In *Proceeding of the 2005 Open International Conference on Modeling & Simulation* 273-279.

Nguyen, H., C. Kästner, and T. Nguyen. 2014. “Exploring Variability-Aware Execution for Testing Plugin-based Web Applications”. In *Proceedings of the 2014 International Conference on Software Engineering* 907-918.

Risco-Martín, J. L., S. Mittal, J. C. Fabero, P. Malagón, and J. L. Ayala. 2016. “Real-time Hardware/Software co-design using Devs-Based Transparent M&S Framework”. In *Proceedings of the 2016 Summer Computer Simulation Conference* 45-52.

Roldán M. L., S. Gonnet, and H. Leone. 2013. “Knowledge Representation of the Software Architecture Design Process based on Situation Calculus”. *Experts Systems* 30(1):34–53.

Shang, H., and G. Wainer. 2006. “A Simulation Algorithm for Dynamic Structure DEVS Modeling”. In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 815-822. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Valderas, P., and V. Pelechano. 2011. “A Survey of Requirements Specification in Model-Driven Development of Web Applications”. *ACM Transactions on the Web* 5(2):10-61.

Wainer, G. 2004. “Modeling and Simulation of Complex Systems with Cell-DEVS”. In *Proceedings of the 2004 Winter Simulation Conference*, edited by R.G. Ingalls, M. D. Rossetti, J. S. Smith and B. A. Peters, 45-56. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wainer G., and P. Mosterman. 2010. *Discrete-Event Modeling and Simulation: Theory and Applications*. CRC Press. Taylor and Francis.

Wallis, M., F. Henskens, and M. Hannaford. 2010. “Expanding the Cloud: A Component-based Architecture to Application Deployment on the Internet”. In *Proceedings of the 2010 International Conference on Cluster, Cloud and Grid Computing* 569-570.

Zeigler, B. P. 1976. *Theory of Modelling and Simulation*. New York: John Wiley and Sons, Inc.

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

Zhang, Qi, L. Cheng, and R. Boutaba. 2010. “Cloud Computing: State-of-the-Art and Research Challenges”. *Journal of Internet Services and Applications* 1(1):7-18.

AUTHOR BIOGRAPHIES

MARÍA JULIA BLAS received her Information Systems Engineering degree from Universidad Tecnológica Nacional in 2014. She has a PhD Research Fellowship from the Consejo Nacional de Investigaciones Científicas y Técnicas and works at Instituto de Desarrollo y Diseño INGAR. Her research interests include discrete-event simulation applied to software products and software quality evaluation using software architectures. Her e-mail address is mariajuliablas@santafe-conicet.gov.ar.

SILVIO GONNET received an Engineering degree in Information Systems from Universidad Tecnológica Nacional in 1998, and obtained his PhD degree in Engineering from Universidad Nacional del Litoral in 2003. He currently holds a Researcher position at Consejo Nacional de Investigaciones Científicas y Técnicas and works at Instituto de Desarrollo y Diseño INGAR. Also, he works as an Assistant Professor at Universidad Tecnológica Nacional. His research interests are models to support design, software architectures, and semantic web. His e-mail address is sgonnet@santafe-conicet.gov.ar.

HORACIO LEONE is a Full Professor at the Department of Information Systems Engineering of Universidad Tecnológica Nacional. He also holds a Researcher position at the Consejo Nacional de Investigaciones Científicas y Técnicas, working at Instituto de Desarrollo y Diseño INGAR. He obtained his PhD degree in Chemical Engineering from Universidad Nacional del Litoral in 1986 and was a Postdoctoral Fellow at the Massachusetts Institute of Technology (MIT). His current research activities focus on software architectures, models for supporting the design process, semantic web applications to supply chain information systems, and enterprise modelling. He has supervised several PhD students. His email address is hleone@santafe-conicet.gov.ar.