Resource-based modeling and simulation of business processes

Andrea d'Ambrogio, Grégory Zacharewicz

Resource-based Modeling and Simulation of Business Processes

Andrea D'Ambrogio
Dept. of Enterprise Engineering
University of Roma TorVergata
Via del Politecnico 1
I-00133 Roma (Italy)
dambro@uniroma2.it

Gregory Zacharewicz
Lab. IMS UMR CNRS 5218
University of Bordeaux
351 Cours de la Libération
33405 Talence cedex (France)
gregory.zacharewicz@u-bordeaux.fr

ABSTRACT
The simulation-based analysis of business processes (BPs) is a key activity at various phases of the BP lifecycle, from the design phase, to predict the process behavior, down to the execution and improvement phases, to recover from possible performance downgrades and/or improve the process performance. The BP analysis is usually carried out taking as input the BP description in a given BP modeling language. This paper specifically addresses BPs described in BPMN (Business Process Model & Notation) and introduces an approach that exploits both model-driven principles and the DEVS (Discrete Event System Specification) formalism to first annotate the BPMN model with the allocation of task resources described in terms of performance and reliability properties and then transform the annotated BPMN model into a DEVS-based model, which can be eventually executed to get the analysis results of interest. The BPMN annotation is carried out by use of PyBPMN, a lightweight BPMN extension that allows business analysts to specify the allocation of task resources and their properties in terms of both time-related attributes and reliability attributes. The paper overviews the proposed approach and gives the details of the DEVS components that are used to model the behavior of the corresponding BPMN primitives.

Author Keywords
Modeling and Simulation, Business Process, BPMN, DEVS, Performance, Reliability

ACM Classification Keywords
I.6.5 Computing Methodologies: SIMULATION AND MODELING—Model Development; C.4 Computer Systems Organization: PERFORMANCE OF SYSTEMS

INTRODUCTION

A *Business Process (BP)* consists of a set of related *tasks* executed by human or automated *resources* to accomplish well-defined goals, such as produce goods or provide services [1].

SummerSim-SCSC 2016 July 24-27, Montreal, Quebec, Canada

Modern BP management approaches strongly recommend the adoption of techniques to concretely support the continuous *BP analysis*, throughout the entire lifecycle, from the initial phases (when the functional and non-functional requirements are specified) down to the final phases (when the performance is monitored and measured at execution time), in order to assess whether or not performance objectives are met and plan appropriate recovery actions when needed.

As argued in [2], effective BP analysis approaches should focus on the use of adequate *modeling and simulation (M&S)* techniques, which enable enterprise management to figure out how to optimize BPs in order to maximize the technical quality and eventually the quality of business.

The adoption of M&S-based approaches is essential in the BP management domain, in which the competitive and dynamic nature of the global marketplace pushes enterprises to enact a continuous effort aimed at the improvement of provided services and goods. In this respect, the use of business process modeling combined with the adoption of simulation-based analysis provides a cost effective, accurate, and rapid way to evaluate alternatives before committing the required effort and resources [3, 4].

On the other hand, the concrete use of simulation-based analysis of BPs is still limited, mainly due to the fact that building simulation models require a non-negligible effort and significant skills [4, 5, 6].

In this context, this paper proposes an automated approach to *build* and *parameterize* simulation models of BPs defined by use of *BPMN (Business Process Model and Notation)*, the standard language for BP specification [7]. The proposed approach adopts model-driven standards and tools, as well as the *DEVS (Discrete Event System Specification)* formalism, to analyze the BP behavior.

The simulation-based analysis of BPs usually focuses on the performance behavior of processes from the efficiency point of view only (e.g., in terms of time-related properties such as throughput or lead time), without taking into account the important issue of process reliability, i.e., the

probability that the BP performs correctly in a given timeframe (often referred to as mission time). The ability to predict the reliability of a BP is instead important to assess the effectiveness of a performance improvement or optimization.

From the reliability point of view, BPMN natively provides constructs to represent issues that affect or alter the BP execution flow (e.g., operation failures, error events, timeout events, etc.), as well as mechanisms to specify actions that have to be carried out to ensure the consistency of the failed BP instance (e.g., compensations, errors handling, etc.). All such failure- and error-related events are similar to exception handlers and are explicitly introduced in the BPMN model by the BP designer.

Differently, in this paper we aim to introduce a reliability analysis that takes into consideration unexpected failures of the resources that execute the process tasks or, in other words, those events that are not natively specified in standard BPMN models and that cause the abnormal interruption of the affected process instance (e.g., the failure – and thus the unavailability – of a resource allocated to a task).

The proposed approach makes use of the DEVS formalism [8] to specify the BP behavior in terms of performance and reliability properties of task resources. On the one hand, the adoption of the DEVS formalism allows to take advantage of the several available tools that can be used to execute (i.e., simulate) the model. On the other hand, building a DEVS model of a given BP specified in BPMN requires a deep knowledge of the formalism, which business analysts are usually not familiar with. Moreover, manually building the DEVS model can be effort- and time-consuming, as well as error prone.

To overcome such limitations, this paper proposes an automated model-driven approach that takes as initial input an annotated BPMN model of the BP under study and yields as output the corresponding DEVS model, ready to be executed. The annotated BPMN model includes the allocation of task resources, which are described in terms of their performance and reliability properties. To this purpose, this work makes use of an extended version of PyBPMN (Performability-enabled BPMN), a lightweight BPMN extension that addresses the specification of performance and reliability properties for a set of resources that may allocated to process tasks [9, 10].

The remainder of this work is structured as follows: the background section summarizes the main concepts at the basis of this paper, specifically the principles of model-driven approaches and the PyBPMN extension. The following sections describe the model-driven approach for generating the DEVS model, the DEVS atomic models of BPMN primitives and an example application of the proposed approach, respectively. Finally, the last section gives some concluding remarks.

BACKGROUND

The next two subsections summarize the principles and the standards introduced in the model-driven engineering field and the PyBPMN extension that is used to annotate BPMN models with the allocation of task resources, as well as with the performance and reliability properties of such resources.

Metamodeling and Model Transformations

The BP specification has been often based on methods that exploit process-related data and text documents in different formats. Such *document-based* manual approach presents natural limitations that have been addressed by the *model-based* approach, which results in many significant advantages, in terms of improved quality, enhanced communication and stakeholder engagement, increased productivity, enhanced knowledge transfer, and reduced risks.

These improvements can be further enhanced by the use of *model-driven* approaches, which increase the level of automation throughout the BP lifecycle by considering models as first-class artifacts. The use of such approaches enables a radical shift in terms of modeling activities, from a strictly contemplative use of models to a more productive and powerful model use.

Metamodeling techniques and automated model transformations are key enabling principles introduced by such approaches in the broader field of *model-driven engineering* [11]. A *metamodel* is a model used to describe a family of models, in other words it is a model that defines the primitives of a modeling language, which is used to specify models at user level. As an example, the BPMN metamodel is the model defining the primitives (i.e., task, gateway, event, etc.) that are instantiated in standard BPMN models. A *model transformation* is the specification of a set of mapping rules that are executed to transform a given model into a different model, which conforms to the same or to a different metamodel.

Various incarnations of model-driven engineering principles have proposed different standards and tools claiming to support model-driven engineering. The approach proposed in this paper makes use of ATL (Atlas Transformation Language) as the language to specify model transformations, which is then executed on top of the Eclipse platform in order to map elements of an input model into elements of the output model [12]. Each model is instantiated from and conforms to a given metamodel, which can be specified either in MOF (Meta Object Facility), the metamodeling language defined by the OMG [13], or in Ecore, the metamodeling language defined by the Eclipse Modeling Framework [14].

PyBPMN-based annotation of BPMN models

The BPMN language does not natively provide any construct to associate performance or reliability properties to process elements.

In order to specify the performance and reliability characterization of BPs, this paper approach makes use of *text annotations*, which provide the required information according to a well-defined syntax.

A BP is a collection of interconnected *activities*, which are executable elements that can be atomic (i.e., *tasks*) or non-atomic (i.e., *sub-processes*). The execution of a process task requires the availability of specific *resources*, i.e., human resources, devices and/or software services.

A BPMN model provides an abstract description of a BP in terms of a set of tasks. The BP design and enactment is then obtained through the allocation of concrete resources to tasks.

The allocation of resources to tasks may lead to the identification of different *configurations*. For instance, a given *service task* could be implemented by different web services, as well as a *manual task* could be performed by different persons. A given BP configuration makes use of a specific web service and a specific person for the service task and the manual task, respectively.

According to this perspective, the BPMN model must be appropriately annotated to specify both the resource allocation and the performance and reliability characterization of such resources. Such annotation is carried out by use of *PyBPMN* (*Performability-enabled BPMN*), a lightweight BPMN extension that addresses the specification of performance and reliability properties of a BP [9, 10].

In order to enable the representation of different configurations implementing the same abstract BP, the PyBPMN metamodel allows business analysts to represent the set of *resources* associated to each BPMN `FlowNode` element. Such element is used in the BPMN metamodel to provide the single source and target elements of a sequence flow that shows the order of elements in a process (as an example, the `Activity` metaclass, which is used to represent the work tasks of a process, is a sub-class of the `FlowNode` metaclass). Each resource is characterized by different performance and reliability properties.

Figure 1 shows the key metaclasses of the PyBPMN metamodel. The `PyElement` is the base abstract metaclass, used to specify workload (`GaWorkloadEvent`), reliability (`DaQualification`) and performance (`PaQualification`) properties. The reader is sent to [boccia11b] for a detailed description of the relevant attributes.

Such properties can be associated either to `FlowNode` elements through a `PyDescriptor` element or to resources being referenced by the `PyDescriptor` element.

The proposed resource modeling is flexible enough to enable a fine grained specification of resource usage. Resources are modeled using the composite pattern, thus enabling the specification of either a simple resource (`PyResource`) or a subsystem (`PySubsystem`), which is composed by any set of simple resources and other subsystems.

The representation of different configurations for a BP is obtained associating alternative resources to the same `FlowNode` element.

The `PyResourceBroker` metaclass has been introduced to enforce the selection of a resource over a set of alternative resources (which can be simple resources or subsystems). For example, if a `FlowNode` element is allocated to two kinds of resources, where one kind (resource A) is known and the other kind can be alternatively implemented by two concrete resources (resources B1 and B2), the `PyDescriptor` element associated to the `FlowNode` element will have two resource references: one to `PyResource` A and one to a `PyResourceBroker` element having `PyResource` B1 and `PyResource` B2 as `alternatives` association ends.

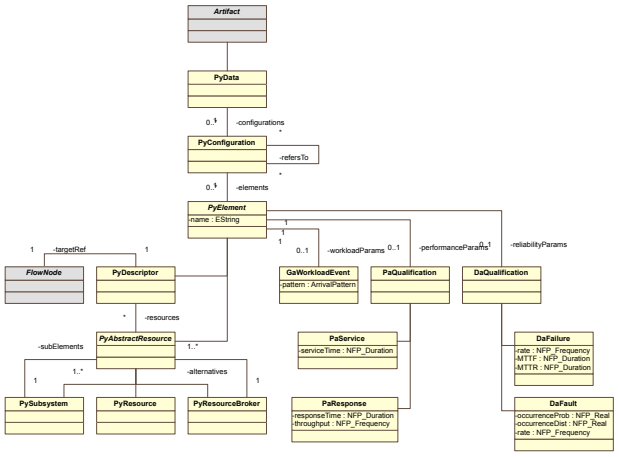

Figure 1. The PyBPMN metamodel.

The definition of resources and their performance and reliability parameters is carried out in the BPMN language using text annotations with a specific syntax.

Such a solution, based on standard BPMN elements such as `TextAnnotation` elements, allows business analysts to specify resource parameters using any BPMN editor.

A `PyResourceBroker` element is instead used to specify alternative allocations, which may be used to identify alternative resources used by a task in case of resource failures.

The resources allocated to a `FlowNode` element are then specified associating the corresponding `TextAnnotation` elements.

According to this allocation mechanism, in case a single resource is allocated to a given task, that resource may be working, thus offering a given performance to the task, or not working, thus implying a failure of the process instance that uses that resource. In case a set of alternative resources

are allocated to a given task, when a resource fails the task may use an alternative resource, which provides different performance properties, and the task (as well as the process instance) fails only if both resources fail.

OVERVIEW OF THE APPROACH

Figure 2 presents the proposed approach, which is based on metamodeling, model mapping and model transformations. Three different levels are identified: Model, MetaModel and MetaMetaModel. The BPMN model is the source model to be transformed, while the DEVS model is the target model resulting from the ATL transformation. BPMN and DEVS models conform to the BPMN 2.0 and DEVS metamodels [15], respectively. The novelty in this mapping, regarding previous results presented in [16], resides in BPMN model is assumed to be annotated by use of the PyBPMN extension described in the background section.

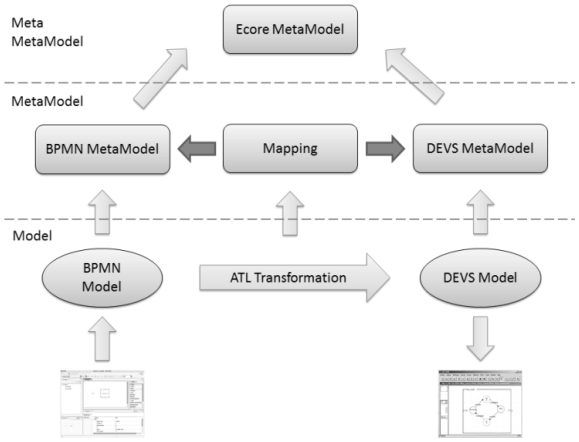

Figure 2. The model transformation approach.

Both metamodels have been specified in Ecore, which is placed at the MetaMetaModel level in Figure 2, to denote the fact the it is used to define the metamodels at the MetaModel level. The mapping that defines the transformation rules to generate DEVS models from annotated BPMN models is specified at the MetaModel level as well. As aforementioned, such a mapping is specified by use of ATL and executed on top of the Eclipse platform.

While the BPMN metamodel is officially released by the OMG (Object Management Group), there is no standard metamodel for the target DEVS metamodel. A synthesis of various proposals for building a DEVS metamodel is proposed in [15]. The transformation from BPMN models to DEVS models has required gathering previous works for setting up a DEVS metamodel, which has been here adapted and simplified. The resulting metamodel, illustrated in Figure 3, conforms to the DEVS specification [8] and has been specified in Ecore.

DEVS models can be of two types, *atomic* and *coupled* models. Each model has a list of input ports and output ports. An atomic model has four main methods: internal transition, external transition, output, and time advance. A coupled model is a decomposition of DEVS models (atomic or coupled) and DEVS coupling. In addition, there are three types of coupling between ports: external input coupling (connections between the input ports of the coupled model and its internal components), external output coupling (connections between the internal components and the output ports of the coupled model) and internal coupling (connections between the internal components).

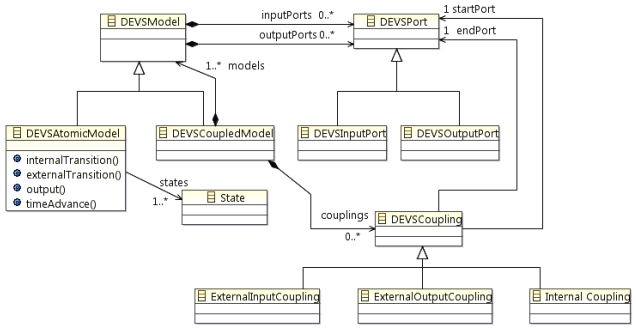

Figure 3. Simplified DEVS metamodel.

The proposed transformation extends previous contributions [16, 17], in which BPMN models are transformed into DEVS and G-DEVS simulation models. Nevertheless, in such contributions resources are not explicitly allocated to tasks, so that they are assumed to be all time available (in other words, resources do not fail).

This paper extends the previous works by introducing a resource allocation mechanism and by explicitly taking into account the failure of resources. Consequently, the DEVS models library for BPMN has been also extended.

The resource allocation mechanism is inspired by works developed in the workflow modeling field [8]. Workflow models make a clear distinction between roles and actors, where a role is a logical abstraction of one or more physical actors. The idea is to introduce this distinction by annotating the BPMN model with resources and brokers, according to the PyBPMN annotation described in the background section. In this respect, resource brokers can be considered as roles that can be played by different actors (i.e., the brokered resources) that possess the required capacity.

MAPPING BPMN COMPONENTS TO DEVS

The role of mapping in model transformation consists in defining links between concepts and relations from source and target metamodels (BPMN and DEVS in this paper case, respectively).

The various types of tasks defined by the BPMN 2.0 specification (e.g.., send and receive tasks, service tasks, etc.), as well as gateways, flows and events are mapped to DEVS atomic models, which are then coupled to define a DEVS model of the overall BPMN process. Next subsections describe both the behavioral description of BPMN tasks and resources to DEVS atomic models and

their coupling. The novelty in the DEVS models with respect to previous works [16, 17] is the DEVS actors behavior, which has been extended to integrate reliability, and a broker model, which has been added to carry out a performance-aware selections of the actor that is available to perform a task.

DEVS atomic model of the "Basic Task" component

The DEVS model of BPMN tasks (Figure 4) proposed in [16] has been extended to integrate the resource allocation mechanism. This model is initialized in a state "free", waiting for a triggering item to be treated (items can be considered as the sequential triggering coming from sequence flows or message flows). After receiving the item, the task performs an allocation request to a resource. It calls the resource broker, or directly the resources connected to the task, to identify an actor with the potential skills to execute the task. If the resource is available, the task can complete the work on the item, according to the resource performance properties, and then release it. If the resource is not available, the item is put on hold and a new allocation request is made to the resource after a given delay.

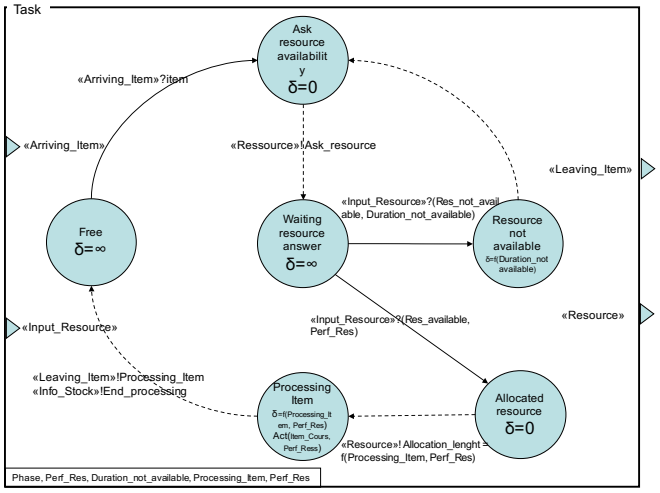

Figure 4. DEVS atomic model of basic task.

DEVS atomic model of the "Resource" component

The DEVS Resource model (Figure 5) has been initially proposed in [17]. This model can be generally described as follows. It is initially set to awaiting an allocation request. When a request is received while the model is in this state, the resource informs the broker or the task of its availability and communicates its performance properties (which are used to determine the task duration). Then, the task model that requested the use of this resource computes the allocation period, according to the PyBPMN annotation, and communicates it back to the resource. Then the resource is set to a busy state for the given amount of time, consistently with the relevant PyBPMN annotation, and therefore cannot, in this state, be allocated to another task. As a result, other tasks performing an allocation request will receive a negative response to their request. When the time allocation has expired, the state model becomes free again and thus available to the allocation task.

DEVS atomic model of the "Resource with Behavior" component

In order to improve the ability of the model to represent real world cases, the resource model has been further extended to integrate the behavior of the resource. As an example, depending on the number of solicitations of the resource a degradation of its performance and reliability properties can be observed. This DEVS atomic model of the resource (Figure 6) is similar to the previous model but adds a local behavior. Such behavior can be defined in this model or synchronized with a central behavior model to observe group effects or compute failures with stochastic approaches at the global level. This possible influence on this model from another role model can be tuned to fit experience gathered by experts for instance. In any case, we can distinguish two types of role models, material resources models and human resources models.

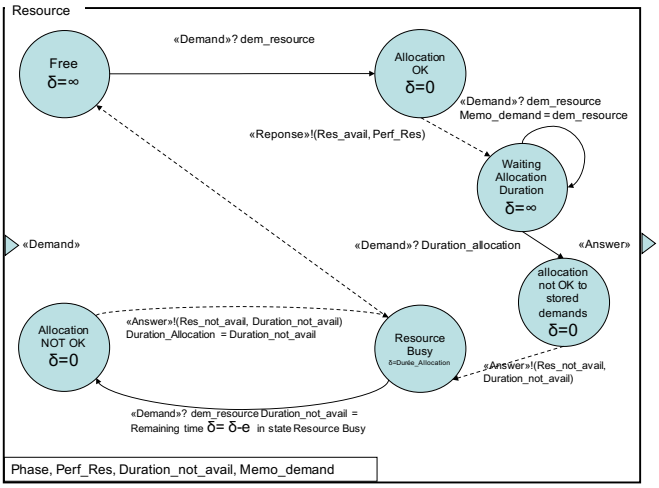

Figure 5. DEVS atomic model of basic resource.

The DEVS model integrating the equipment and machine resource behavior, not further detailed in this paper, is defined from technical data sheets. These documents are typically released by resource manufacturers and usually provide reliability and availability properties of these resources in terms of MTTF (mean time to failure) and MTTR (mean time to repair).

The DEVS model of human behavior has been based on psychological and physiological factors. Specifically, factors such as emotions, stress, and fatigue may be taken into account. These models are based on the work described in [18], which defines stereotyped human behavior using DEVS at the individual level, and in [19], which defines social influence using DEVS at the group level. They have both defined DEVS models able to represent, even highly simplified, dynamics of human behavior. Such models can

provide valuable information to control and anticipate the operational safety of the modeled BP.

These behavioral models contribute to yield a more realistic modeling and simulation of the overall business process. The standard resource model (without failure behavior) is available by default. Then, according to data from PyBPMN annotation, it can be changed by the resource model with behavior. In any case, it interacts with the broker model to verify its availability and the duration of allocation. In case of availability, the resource informs the broker or directly the task about its current performance and reliability capabilities, which are used to compute the task duration and performance. The resource can have its performance decreased due to simulation computable factors including, e.g., fatigue or stress due to number of solicitations. If the resource is not available, the broker can manage a queue to store the demand and wake it up when it becomes available again.

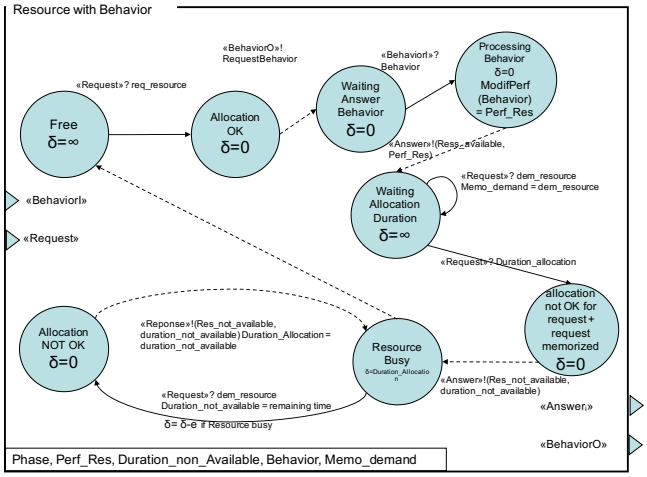

Figure 6. DEVS atomic model of resource with behavior.

DEVS coupled model of a Business Process

The DEVS atomic models described in previous sections are coupled to yield the model of the overall BP specified by use of a BPMN model. This requires applying a set of rules for coupling the components.

As aforementioned, the model transformation has been specified in ATL, which provides the language primitives to specify mapping rules by use of a hybrid approach (both declarative and imperative). Such rules can be informally outlined as follows:

- **Rule 1**: Each BPMN model must contain a Start event and an End Event models.
- **Rule 2**: Each task model is connected upstream to a connector or a task.
- **Rule 3**: Each task model is connected downstream to a connector or a task.
- **Rule 4**: Each task model is connected to a resource or resource broker model.
- **Rule 5**: Each task model is connected to (at least) a resource model.
- **Rule 6**: Each connector (gateway or event) model is connected upstream and downstream to one or more task models.

The next section illustrates an example application of the proposed approach. Since the focus of this paper is on model building, the example application will be limited to the automated generation of the DEVS coupled model from of a given BPMN model. The so obtained model can then be executed onto one of the several DEVS implementation tools.

APPLICATION CASE

The proposed DEVS model building approach is illustrated by use of an example application to the BPMN model shown in Figure 7, which consists of three tasks, a start node, an end node and two exclusive gateway (X-Or) connectors.

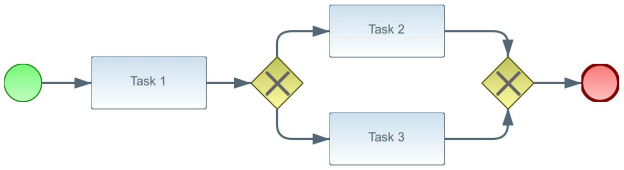

Figure 7. Example BPMN model.

As defined above, each DEVS model of a task (Task 1, Task 2 and Task 3) must be associated with the DEVS model of the resource or resource broker enabling it to execute the task. The allocation of tasks to resources is carried out by use of PyBPMN annotations, specified according to the syntax described in the background section (not shown in Figure 7 for the sake of readability). As an example, the following text annotation specifies a resource with name `resource1`, service time of 250 milliseconds and MTTF of 50000 hours:

```
<<PyResource>>{
  name=resource1,
  performanceParams=(<<PaService>>{serviceTime=
                    (value=250, unit=ms)}),
  reliabilityParams=(<<DaQualification>>{MTTF=
                    (value=50000, unit=hours)})
}
```

Alternative resources may be specified by use of resource brokers. As an example, the following text annotation specifies a resource broker for two alternative resources (`resource1` and `resource2`):

```
<<PyResourceBroker>>{alternatives={resource1,
                     resource2}}
```

This model illustrates different coupling situations that can occur in a BPMN model. A task can be connected directly to resources or through a broker that will identify at run time the best resource according to expected or required levels of performance and reliability.

The transformed DEVS coupled model is presented in Figure 8, which shows how the tasks have been connected to specified resources or brokers, according to PyBPMN annotations. It distinguishes, in particular, that the simplest case is the direct connection between task and resource that occurs between Task 3 and Resource 3. In the case of Task 1 and Task 2, a broker is used to select the best available resource according to performance and reliability criteria, as the ones described in [10], which presents an algorithm based on *performability*, a joint measure of performance and reliability. The coupled model also contains input and output atomic models to generate and collect events.

The behavior of the tasks and resources is open to reconfiguration, according to the requirements coming from the BPMN annotations. The execution of the obtained DEVS model allows evaluating alternative resource allocations, so as to choose the best fitting without modifying the structure of the input BPMN model.

The coupled model includes the DEVS models of start/end event nodes, which are used to initialize and forward the outgoing events, respectively. Outgoing events are eventually coupled to an additional model that can classify, process and analyze the simulation results. The analysis of such results allows one to evaluate the key performance indicators of the considered BP, in terms of efficiency, reliability and other aggregated metrics.

The DEVS simulation model allows bridging the gap between the BPMN model, which is commonly a static view of the process, and the execution over time of the model according to the performance and reliability properties offered by resources allocated to tasks.

PERSPECTIVE ON INTEROPERABILITY

This work is going to open the possibility of considering *interoperability* as an additional parameter for the selection of resources to be allocated. Indeed, a task sequence triggering is trivial but the transmission of an item or information to the next task can face interoperability issues. Thanks to the definition of an interoperability capacity (indicator), we plan to discriminate resources considering their capacity to respond to the information to be processed or the service to be provided. At the BPNM level such resources will still belong to the same pool but they will be evaluated to obtain the required levels of reliability and efficiency when executing the task. The selected resource will be the one that, according to criteria, is estimated to better address interoperability issues.

In more detail, the transmission of messages between independent tasks and resource models will have to overcome different categories of interoperability barriers, so as to enact a correct flow of messages. In order to include such interoperability-oriented characterization, the model will be extended to represent the effort required and the category of interoperability (e.g., syntactic, semantic,

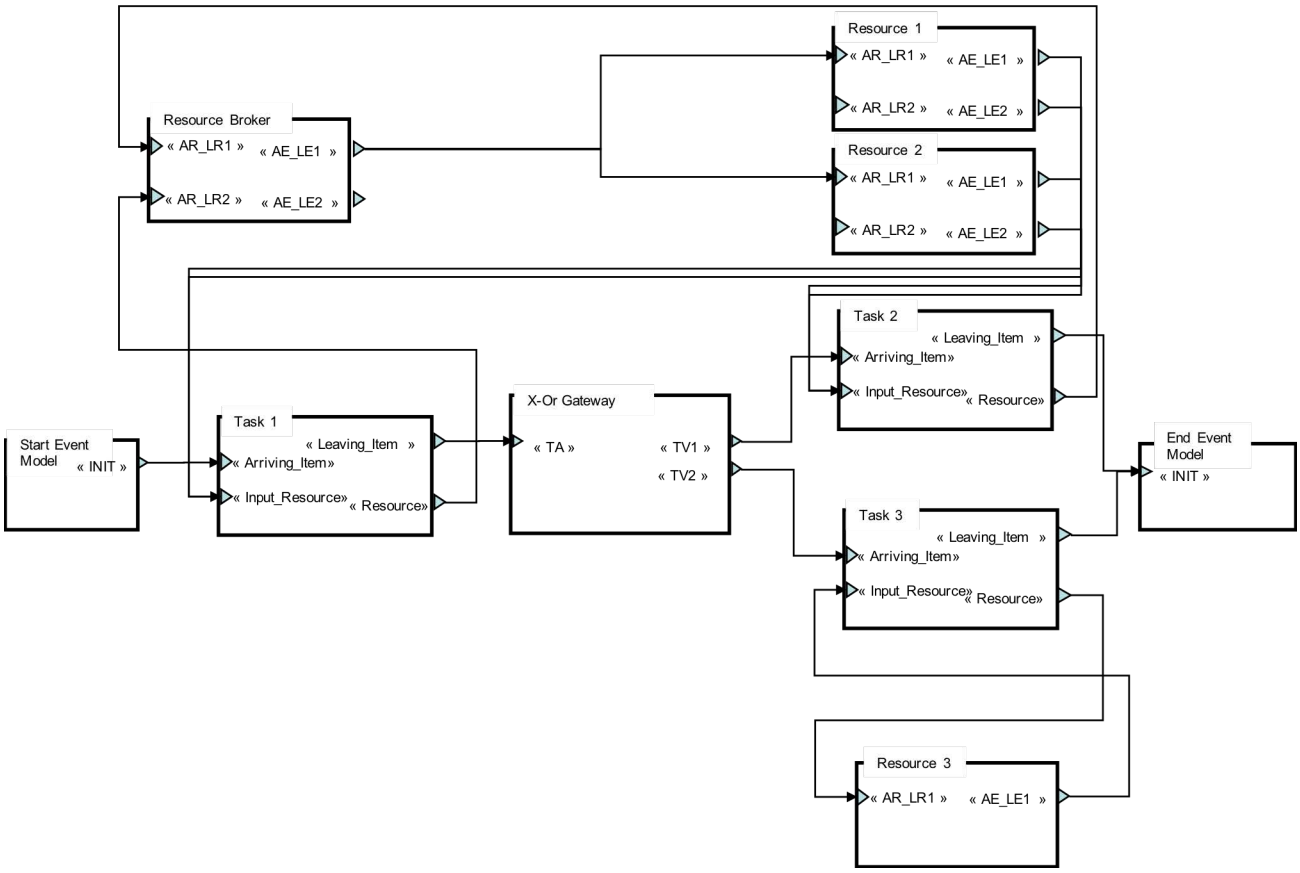

Figure 8. DEVS coupled model of the example BP.

technological) that can satisfy the requirement of correct and unambiguous communication.

CONCLUSIONS

This paper has introduced a model-driven approach to generate DEVS-based simulation models from annotated BPMN models. The BPMN annotation is carried out by use of the PyBPMN extension, which allows business analysts to specify the allocation of task resources, as well as their performance and reliability properties. The annotated BPMN model is then taken as input by a model transformation that yields as output the corresponding DEVS model, ready to be executed by use of a DEVS simulator. The proposed approach extends previous works that didn't consider the resource allocation mechanism and the failure of resources. The paper has illustrated the mapping of BPMN elements to DEVS atomic models, which are then coupled according to the process control flow logic to get the final DEVS coupled model. Work is in progress to complete the specification of the mapping rules, which currently have been developed for basic BPMN elements only.

ACKNOWLEDGMENTS

The second author gratefully acknowledges the grant from the MSEE (Manufacturing Service Ecosystem) EU project.

REFERENCES

1. Weske, M., Business Process Management second edition, Springer-Verlag, 2012.
2. Van Der Aalst, W. M., Business Process Management: a Comprehensive Survey. ISRN Software Engineering, 2013.
3. Tumay, K., Business process simulation". In Proceedings of the 1996 Winter Simulation Conference, edited by J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 93–98, 1996.
4. Van der Aalst, W., J. Nakatumba, A. Rozinat, Russell, N., Business Process Simulation: How to get it right?, in Handbook on Business Process Management, edited by J. vom Brocke and M. Rosemann, 317–342: Springer-Verlag, 2010.
5. Hook, G., Business Process Modeling and simulation, Proceedings of the 2011 Winter Simulation Conference, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. C. Fu, 773–778, 2011.
6. Kamrani, F., R. Ayani, Karimson, A., "Optimizing a Business Process Model by Using Simulation", Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation, 1–8, 2010.
7. Object Management Group, Business Process Model and Notation (BPMN), version 2.0, 2011.
8. Zeigler, Bernard P., Herbert Praehofer, and Tag Gon Kim. Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. Academic press, 2000.
9. Bocciarelli, P., D'Ambrogio, A., A BPMN Extension for Modeling Non Functional Properties of Business Processes, Proceedings of the 2011 Symposium on Theory of Modeling and Simulation, 160–168: Society for Computer Simulation International, 2011.
10. Bocciarelli, P., D'Ambrogio, A., A Model-driven Method for Enacting the Design-time QoS Analysis of Business Processes, Software & Systems Modeling 13:573–598, 2014.
11. Atkinson, C., Kuhne, T., Model-driven development: a metamodeling foundation, IEEE Software, 20(5), pp. 36-41, 2003.
12. ATL/User Guide – The ATL Language, available at http://wiki.eclipse.org/ATL (last accessed on April 2016).
13. Object Management Group, Meta Object Facility (MOF) Specification, version 2.0, 2004.
14. Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. Eclipse Modeling Framework. Addison Wesley Professional, 2004.
15. Garredu S., Vittori E., Santucci J.F., Bisgambiglia P. A Meta-Model for DEVS - Designed following Model Driven Engineering Specifications. In Proceedings of SIMULTECH, pp. 152-157, 2012.
16. Bazoun, B., Ribault, J., Zacharewicz, G., Ducq Y., Boyer H., SLMToolBox: Enterprise service process modelling and simulation by coupling DEVS and services workflow, International Journal of Simulation and Process Modeling, In Press. 2016.
17. Zacharewicz, G., Frydman, C., Giambiasi, G., G-DEVS/HLA environment for distributed simulations of workflows, Simulation 84 (5), 197-213, 2008.
18. Seck, M., Giambiasi, N., Frydman, C., Baati, L., DEVS for human behavior modeling in CGFs. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, 4(3), 196-228, 2007.
19. Bouanan, Y., Forestier, M., Ribault, J., Zacharewicz, G., Vallespir, B.; Devs-based framework for message dissemination in multi-layer networks, 27th European Modeling and Simulation Symposium, EMSS, Bergeggi; Italy Pages 416-424, 2015.
20. Hollingsworth, D., Workflow Management Coalition: the Workflow Reference Model. WfMC document Number TC00-1003, issue 1.1. January 1995.