Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

BPMN-BASED BUSINESS PROCESS MODELING AND SIMULATION

Paolo Bocciarelli
Andrea D'Ambrogio
Andrea Giglio
Emiliano Paglia

Department of Enterprise Engineering
University of Rome Tor Vergata
Viale del Politecnico 1
Rome, 00133, ITALY

ABSTRACT

A business process (BP) can be defined a set of tasks that are coordinately performed by an organization to achieve a business goal. M&S (Modeling & Simulation) techniques are widely and effectively adopted for BP analysis. A BP M&S approach is typically carried out by first building a simulation model (from a conceptual model of the BP under analysis), and then producing the software implementation of the simulation model, so to eventually execute the simulation code and get the results of interest. The standard language currently used to define BP models is the OMGs BPMN (Business Process Model and Notation). This paper presents a BPMN-based M&S approach that introduces a BPMN extension to specify BP simulation models as annotated BPMN models, and a domain-specific BP simulation language to specify and execute simulation model implementations, which can be seamlessly derived from annotated BPMN models by use of automated model transformations.

1 INTRODUCTION

In recent years, several organizations have shifted towards a massive adoption of process management methodologies and tools, in order to improve the maturity level of their operational processes. The term *Business Process (BP)* is used to define a set of activities that are coordinately performed in an organizational and technical environment to achieve a business goal (Weske 2012). In this context, *Business Process Management (BPM)* is referred to as a comprehensive discipline that makes use of methods, techniques and tools to support the design, analysis, enactment and diagnosis of operational BPs (van der Aalst 2013).

The BP design and analysis activities of a BPM effort strongly rely on the use of process modeling and simulation techniques. The combined use of process modeling and simulation techniques provides an effective approach to analyze BPs and evaluate design alternatives before committing the necessary resources and effort.

Several formalisms, or languages, have been introduced for the specification of BP models, such as flow charts, activity diagrams, petri nets and event-driven process chains. Recently, the Object Management Group has introduced the Business Process Model and Notation (BPMN) (OMG 2019a), a graphical language for the BP representation that has rapidly become a de-facto standard in the BPM area, due to its rich and easy to understand notation. BPMN allows business analysts to specify abstract models of BPs that are then mapped to the execution languages of BPM systems, such as the Business Process Execution Language (BPEL), a standard for Web Services-based BP orchestration (OASIS 2019).

The use of simulation techniques to analyze BPs specified as BPMN models requires first the derivation of a simulation model from the BPMN model, so to provide the necessary details (performance parameters,

execution resources, expected workload, etc.) that are missing in the BPMN model but required to make the model executable. The obtained simulation model is then implemented into a software program, which is eventually run to get the simulation results (performance indicators) of interest.

The concrete use of simulation-based BP analysis is still somehow limited, mainly due to the fact that the simulation model building activity, as well as the implementation and execution of simulation models, require a non-negligible effort and significant skills (van der Aalst et al. 2010; Kamrani et al. 2010; Hook 2011).

This paper presents a BPMN-based M&S approach that introduces a BPMN extension, named *PyBPMN*, to annotate a BPMN model with what is necessary to specify the corresponding BP simulation model, such as performance properties, resources, workloads, etc. The extension has been carried out according to a lightweight profiling mechanism, not to alter the content and validity of the original BPMN model.

In addition, the approach exploits a Java-based domain-specific BP simulation language, named *eBPMN*, which is used to implement BP simulation models specified in PyBPMN. The eBPMN code, ready to be executed, can be seamlessly derived from PyBPMN models by use of automated model transformations.

The rest of this paper is organized as follows: Section 2 deals with BP modeling, presenting both the BPMN standard and its proposed extension (i.e., PyBPMN). Section 3 presents BP simulation techniques and introduces the proposed domain-specific BP simulation language (i.e., eBPMN). Section 4 describes the automated generation of eBPMN-based simulation code, as well as the enabling tool-chain. Section 5 illustrates an example application of the proposed contribution and finally Section 6 gives concluding remarks.

2 BUSINESS PROCESS MODELING

Modeling is a way to manage complexity. A *model* represents a given subject, with respect to a particular scope, by abstracting from details that are irrelevant for that scope. Two main scopes for process modeling can be identified in BPM (Desel and Erwin 2000; Dumas et al. 2013):

- *Organizational design*: the model is used for a complete understanding of the BP during its lifetime, from communication with stakeholders to BP evaluation and improvement. The model is intuitive and generally represented with a graphical notation;
- *Application development*: the model is more detailed and thus includes all the technical information required for the implementation and automation of the BP.

Among all the formalisms and languages defined for the specification of BP models, the Business Process Model and Notation (BPMN) is currently the most widely used (Weske 2012).

2.1 BPMN

The Business Process Model and Notation (BPMN) (OMG 2019a) is a standard graphical notation for the representation of BPs that is easily readable by all the actors involved in BPM. Although BPMN allows to effectively represent a BP at different levels of abstraction, it is more used in the early stages (analysis and design) of the process lifecycle.

BPMN provides three basic types of diagram for the graphical description of BPs:

- Processes (Orchestration);
- Choreographies;
- Collaborations, which can include Processes and/or Choreographies.

The elements of BPMN diagram are divided in four categories as shown in Figure 1: *Flow Objects*, *Artifacts*, *Connecting Objects* and *Swimlanes*.

Flow Objects are the fundamental elements of BPMN and include *Events*, *Activities* and *Gateways*:

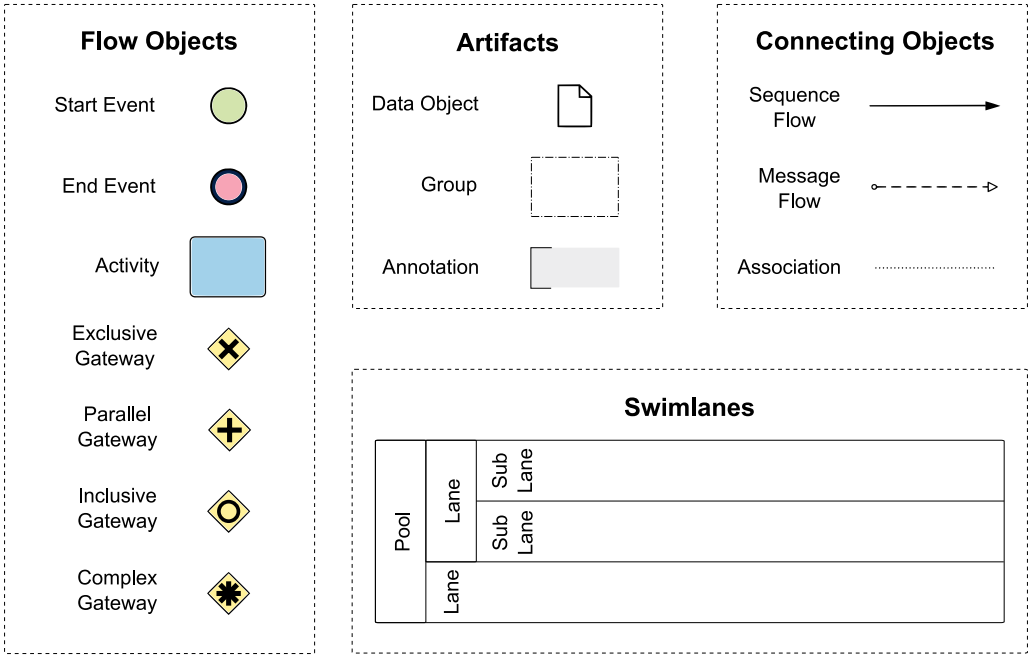

Figure 1: BPMN elements.

- *Events*: something that could happen and that is relevant for the BP. Events are important because they affect the control flow of the BP;
- *Activities*: units of work carried out during the BP (atomic activities are denoted as *Tasks*);
- *Gateways*: decision points that alter the sequential flow between *Flow Objects*. A gateway can be diverging or converging if it splits or merges the execution flow, respectively.

Artifacts are used to specify additional information and documentation about the BP. Among *Artifacts*, *Text Annotations* are used to provide information in the model as free text.

The participants in the BP and their activities can be organized using *Pools* and *Lanes*.

Flow Objects, *Artifacts* and *Swimlanes* can be connected with each other using the following *Connecting Objects*:

- *Sequence Flow*, to connect flow objects in sequential order within a pool;
- *Message Flow*, to exchange messages between participants depicted as *Pools*;
- *Association*, to link information (e.g., *Artifacts*) to *Flow Objects*.

The semantic behavior of the process model is specified through the *token* concept. In particular, the behavior of each BPMN element is defined specifying how it interacts with an incoming token. The token is an abstract element that is created by the start event of the BP, passed through the elements of the process according to a given sequence flow, and finally destroyed at an end event occurrence.

2.2 PyBPMN

The BPMN language has some limitations with respect to the specification of non-functional properties of the BP. To address such issues, several authors have extended the standard BPMN definition.

PyBPMN (Performability-enabled BPMN) is a BPMN extension that addresses the specification of performance and reliability properties of BPs (Bocciarelli and D'Ambrogio 2011a; Bocciarelli and D'Ambrogio 2011b; Bocciarelli and D'Ambrogio 2014; Bocciarelli et al. 2014; D'Ambrogio et al. 2016).

The PyBPMN extension addresses four main areas of non-functional properties:

- **Workload**, to model the workload for the tasks in the process (class `GaWorkloadEvent`);
- **Performance**, to specify the performance properties, i.e., efficiency-related properties such as the service time, associated to single tasks (class `PaQualification`);
- **Reliability**, to express the reliability properties of the resources used by tasks to carry out the work requests (class `DaQualification`);
- **Resource management**, to specify the resources involved in the BP. PyBPMN allows users to specify atomic resources (`PyPerformer`), groups of resources (`PySubsystem`) or sets of alternative resources (`PyBroker`).

2.3 Performance Characterization

PyBPMN introduces the following additional classes for performance characterization:

- `PaQualification`: base abstract class that can be specialized as `PaService` or `PaResponse`;
- `PaService`: specifies the service demand in terms of the following attribute:
  - `serviceTime`: time demand required for the execution of a single request;
- `PaResponse`: specifies the performance as seen from an external user in terms of the following attributes:
  - `throughput`: requests accomplished per time unit;
  - `responseTime`: time spent for the execution of the request.

2.4 Reliability Characterization

PyBPMN introduces the following additional classes for reliability characterization:

- `DaQualification`: base abstract class that can be specialized as `DaFault` or `DaFailure`;
- `DaFault`: specifies fault-related properties using the following attributes:
  - `rate`: fault occurrence rate;
  - `occurrenceProb`: probability of a fault occurrence (time independent);
  - `occurrenceDist`: probability distribution of a fault occurrence in a time interval (time dependent);
- `DaFailure`: specifies failure-related properties using the following attributes:
  - `rate`: failure occurrence rate;
  - `MTTF`: mean time to failure;
  - `MTTR`: mean time to repair.

2.5 Resource Characterization

PyBPMN provides a resources definition which is complementary to that of the standard BPMN. In BPMN, a resource is an abstract role involved in an activity, whereas the PyBPMN resource definition allows to specify, at design time, the real entities that perform the activities, along with their non-functional properties.

A PyBPMN resource can model a human worker, an equipment, a functional division of an organization, an autonomous system, a web service, or any other entity which can be used to carry out a service request.

PyBPMN introduces the following main classes for resources definition:

- `PyRequest`: wrapper class used to specify the fraction of the resource capacity that is required to execute the service;
- `PyBaseResource`: abstract base class that can be specialized as `PyPerformer`, `PyBroker` and `PySubsystem`;
- `PyPerformer`: actual work performer that executes the service requests;

- `PyBroker`: complex resource that manages a set of candidate resources and delegates to one of them the execution of the service request. The resource selection is dynamic: if the currently selected resource is no more available (e.g., due to a failure), the broker dispatches the service request to another available resource (if existing);
- `PySubsystem`: complex resource defined in terms of its elementary components (a set of correlated resources that are all required to accomplish the service request). Differently from the `PyBroker`, the `PySubsystem` sequentially dispatches the service request to all of the resources composing the subsystem;
- `PyDescriptor`: used to associate the PyBPMN resources to standard BPMN elements.

Figure 2 shows the relations between BPMN elements and PyBPMN classes.

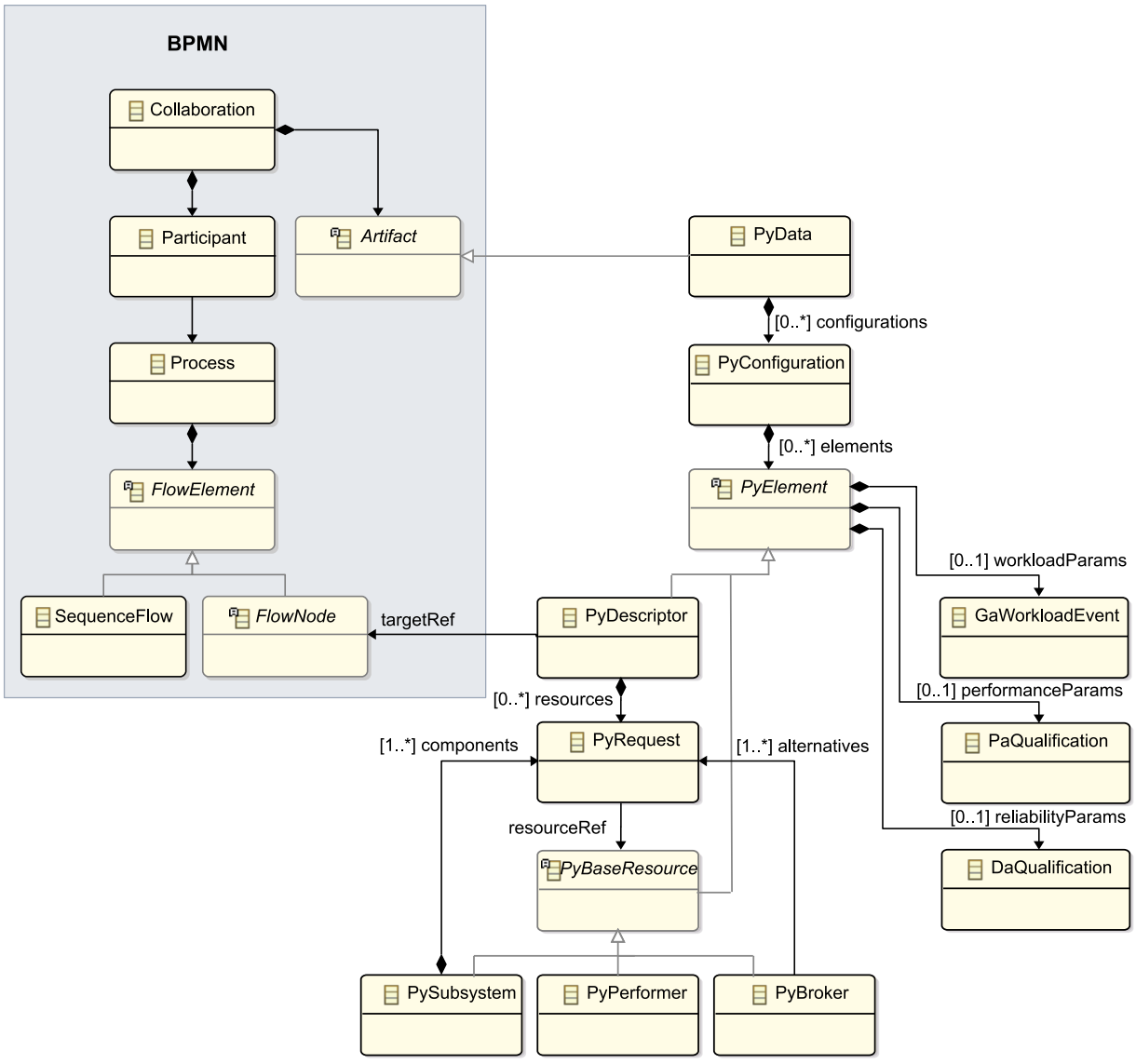

Figure 2: Relations between BPMN and PyBPMN classes.

3 BUSINESS PROCESS SIMULATION

Business Process Simulation (BPS) is considered as the most popular and used technique for quantitative analysis of process models (Dumas et al. 2013). BPS uses a simulation engine to generate a large number of BP instances and to gathers data during their execution.

BPS has several advantages over other analysis methods:

- It allows to analyze complex processes, thanks to its flexibility;
- It allows to analyze several indicators at the same time such as service times, waiting times and resources utilization;
- It can be used for *what-if* analysis, in order to evaluate different BP scenarios at design time before moving to implementation.

The simulation engine can generally be programmed using:

- A language that provides simulation-specific primitives;
- A simulation package, or tool, that provides several domain-specific building blocks that are used to produce and execute the simulation model.

The simulation language approach provides more flexibility but requires more skills and effort. The simulation tool approach is less flexible but more user-friendly and generally provides a graphical interface for the definition of the simulation model.

However, BPS also shows some drawbacks:

- The definition of the simulation model could require programming skills;
- The soundness of simulation results depends on the correctness and completeness of the BP model;
- The accuracy of the results depends on the number of simulation runs, so to easily lead to time-consuming and computation-intensive simulation executions.

3.1 eBPMN

The eBPMN language is a domain-specific simulation language that has been defined to conform to the execution semantics of the BPMN 2.0 specification. Indeed, the eBPMN language simulates the BP execution implementing the token concept defined in the original BPMN specification. Moreover, eBPMN provides the simulation of the resources behavior and non-functional properties defined by the PyBPMN extension. The eBPMN language allows users to simulate either a single process or a collaboration of processes interacting with each other by use of `MessageFlow` elements.

Currently, eBPMN implements only a portion of the entire set of BPMN 2.0 elements, according to the following constraints:

- Each `Pool` must include a `Start` event element (no implicit start allowed);
- Choreographies are not supported;
- Each `FlowNode` must have one incoming `SequenceFlow` and one outgoing `SequenceFlow` (multiple flows require an explicit `Gateway` element);
- `Complex Gateways` are not currently supported.

The eBPMN language exploits the SimArch layered software architecture, which provides a framework for transparently executing discrete-event simulations in either local or distributed simulation environments (Gianni et al. 2008).

4 MODELING AND SIMULATION TOOLCHAIN

An effective approach to bridge the gap between the process model and the simulation model is the one based on model-driven techniques for the automated generation of the simulation code (Bocciarelli et al. 2014; Bocciarelli et al. 2014). Figure 3 shows the process (in BPMN format) that allows to automatically obtain the simulation code from the initial BP definition, so to verify if the requirements set at specification time can be satisfied. In case of requirements not satisfied, the process can be iterated by addressing alternative scenarios in terms of process flow or resources involved in the BP.

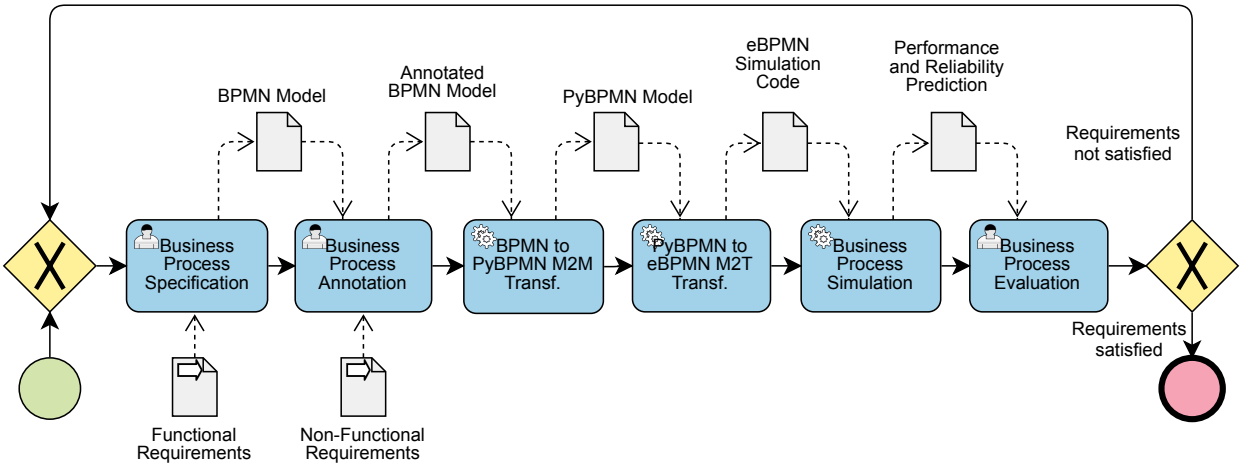

Figure 3: Process for model-driven BP evaluation.

In particular, the proposed approach goes through the following steps:

- **Business Process Specification**: the analyst specifies the BP according to the *functional requirements*. The result is a *BPMN model*;
- **Business Process Annotation**: the *BPMN model* is enriched with textual annotations to specify *non-functional requirements* (workload, performance, reliability, and resources), thus obtaining an *annotated BPMN model*;
- **BPMN to PyBPMN Model-to-Model Transformation**: the *annotated BPMN model* is automatically transformed into the corresponding *PyBPMN model*;
- **PyBPMN to eBPMN Model-to-Text Transformation**: the *PyBPMN model* is automatically transformed into the *eBPMN simulation code*;
- **Business Process Simulation**: the *eBPMN simulation code* is executed by the analyst, thus obtaining a *performance and reliability prediction* of the BP behavior;
- **Business Process Evaluation**: the analyst compares the *performance and reliability prediction* with the initial requirements to verify if they are satisfied.

Once requirements are satisfied, the BP model can be actually implemented, executed and monitored considering the configuration, in terms of process flow and resources, specified for the simulation. Moreover, the method can also be used to improve the behavior of an existing BP or to quickly adapt the BP to environmental changes or performance downgrades.

4.1 Business Process Annotation

The original BPMN model is annotated by using `TextAnnotation` elements having a specific syntax, in order to automatically obtain the PyBPMN model.

A `TextAnnotation` that includes a valid PyBPMN syntax is said to be a `PyAnnotation`. Each `PyAnnotation` contains one or more definitions (i.e., `PyDefinition`) that conform to the following EBNF (Extended Backus-Naur Form) formal syntax (Scowen 1993):

⟨*PyDefinition*⟩ ::= ⟨*PyElement*⟩ '{' ⟨*PyParamList*⟩ '}'

⟨*PyElement*⟩ ::= '<<PyDescriptor>>'
| '<<PyPerformer>>'
| '<<PyBroker>>'
| '<<PySubsystem>>'

⟨*PyParamList*⟩ ::= ⟨*String*⟩ '=' ⟨*Value*⟩ (',' ⟨*String*⟩ '=' ⟨*Value*⟩)*

⟨*Value*⟩ ::= ⟨*Literal*⟩ | ⟨*ComplexValue*⟩ | ⟨*CollectionValue*⟩

⟨*Literal*⟩ ::= ⟨*String*⟩ | ⟨*Number*⟩

⟨*ComplexValue*⟩ ::= '(' ⟨*String*⟩ '=' ⟨*Literal*⟩ (',' ⟨*String*⟩ '=' ⟨*Literal*⟩)* ')'

⟨*CollectionValue*⟩ ::= '(' ⟨*Literal*⟩ (',' ⟨*Literal*⟩)* ')'

In order to clarify the use of the textual annotation used for the specification of the PyBPMN elements, some example definitions are here provided.

To define a `PyPerformer` with name `Perf_1` and a `serviceTime` of 350 ms, the following textual annotation is used:

```
<<PyPerformer>> {
    name = Perf_1,
    serviceTime = (value=350, unit=ms)
}
```

The specification of the resources used by a BPMN task is carried out by using a `PyDescriptor` element in a `TextAnnotation` associated to the BPMN task:

```
<<PyDescriptor>> {
    resources = (Perf_1)
}
```

4.2 Model-to-Model Transformation

The automatic generation of the PyBPMN model is carried out by using a model-to-model transformation (OMG 2019b), named `BPMN_to_PyBPMN` transformation, which takes a BPMN model as input and generates a PyBPMN model as output.

The transformation exploits the textual annotations that conform to the PyBPMN syntax, in order to generate the corresponding PyBPMN elements in the target model.

4.3 Model-to-Text Transformation

The simulation code is then automatically generated through a model-to-text transformation (OMG 2019c), named `PyBPMN_to_eBPMN`, that takes a PyBPMN model as input and produces the corresponding eBPMN simulation code as output.

For each element in the PyBPMN model, the transformation customizes a code snippet template with the values of the properties that are specified in the PyBPMN model.

4.4 Eclipse Plugins

The aforementioned transformations for the automatic generation of the PyBPMN model and the eBPMN simulation code have been implemented as a set of Eclipse plugins, as follows (Eclipse Foundation 2019b):

- a plugin implementing the *PyBPMN conceptual model*;
- a plugin implementing the *BPMN to PyBPMN* model-to-model transformation;
- a plugin implementing the *PyBPMN to eBPMN* model-to-text transformation;
- a plugin implementing the *Eclipse PyBPMN contextual menu*.

The Eclipse platform also includes the BPMN modeler (Eclipse Foundation 2019a) and the simulation engine, so to provide a complete toolchain that eases the production of the simulation code from the BP model. When the user executes a right click on the icon of a BPMN model in the Eclipse workspace, the PyBPMN contextual menu is shown as depicted in Figure 4. In the contextual menu, a sub-menu named **PyBPMN** is available with two commands:

- **BPMN to PyBPMN**, to execute the model-to-model transformation on the selected BPMN model to produce the corresponding PyBPMN model. The resulting PyBPMN model can be explored and modified using the default tree editor;
- **BPMN to eBPMN**, to sequentially execute both the model-to-model and the model-to-text transformations. This can be useful to generate the simulation code with a one click operation, without the explicit generation of the intermediate PyBPMN model, as illustrated in Figure 5.

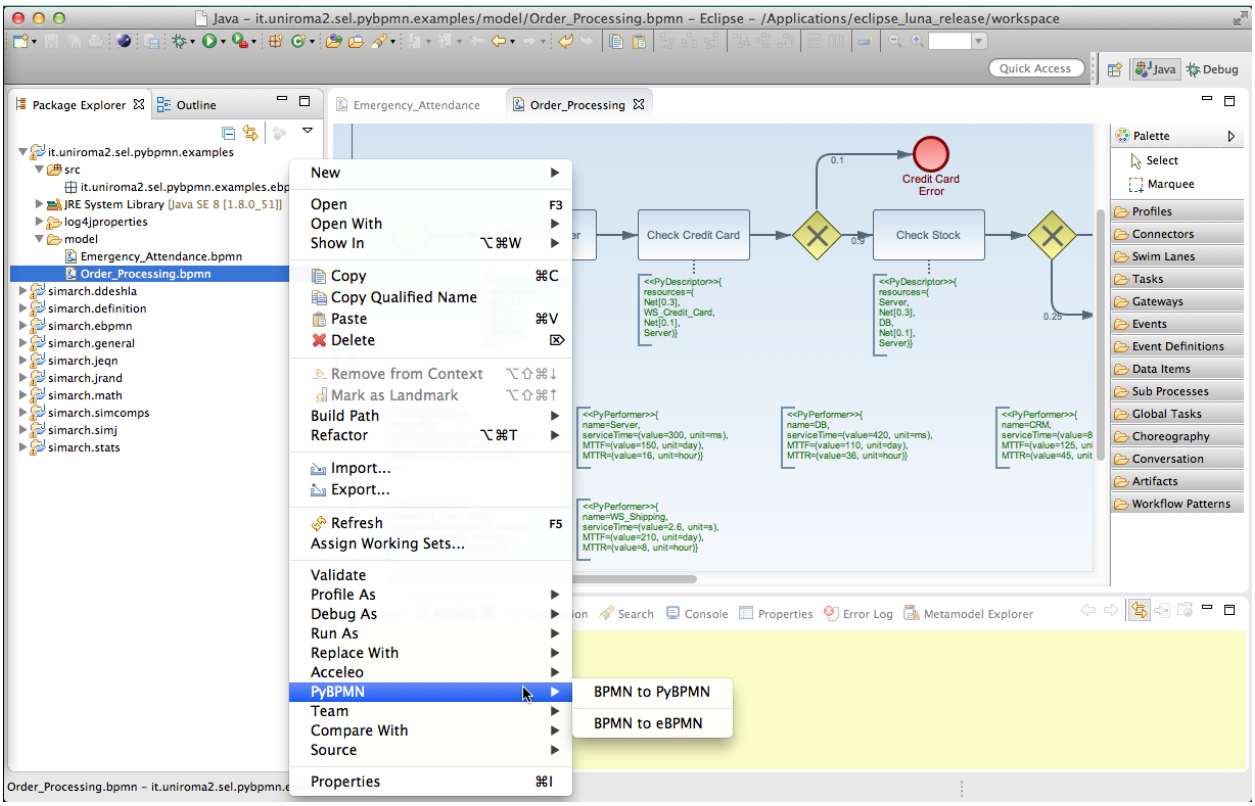

Figure 4: Contextual menu with the PyBPMN commands to apply the model transformations.

The PyBPMN contextual menu is model-aware and only shows the transformations that can be effectively applied on the selected model. Thus, if the user executes a right click on a PyBPMN model, the contextual menu only shows the PyBPMN to eBPMN transformation.

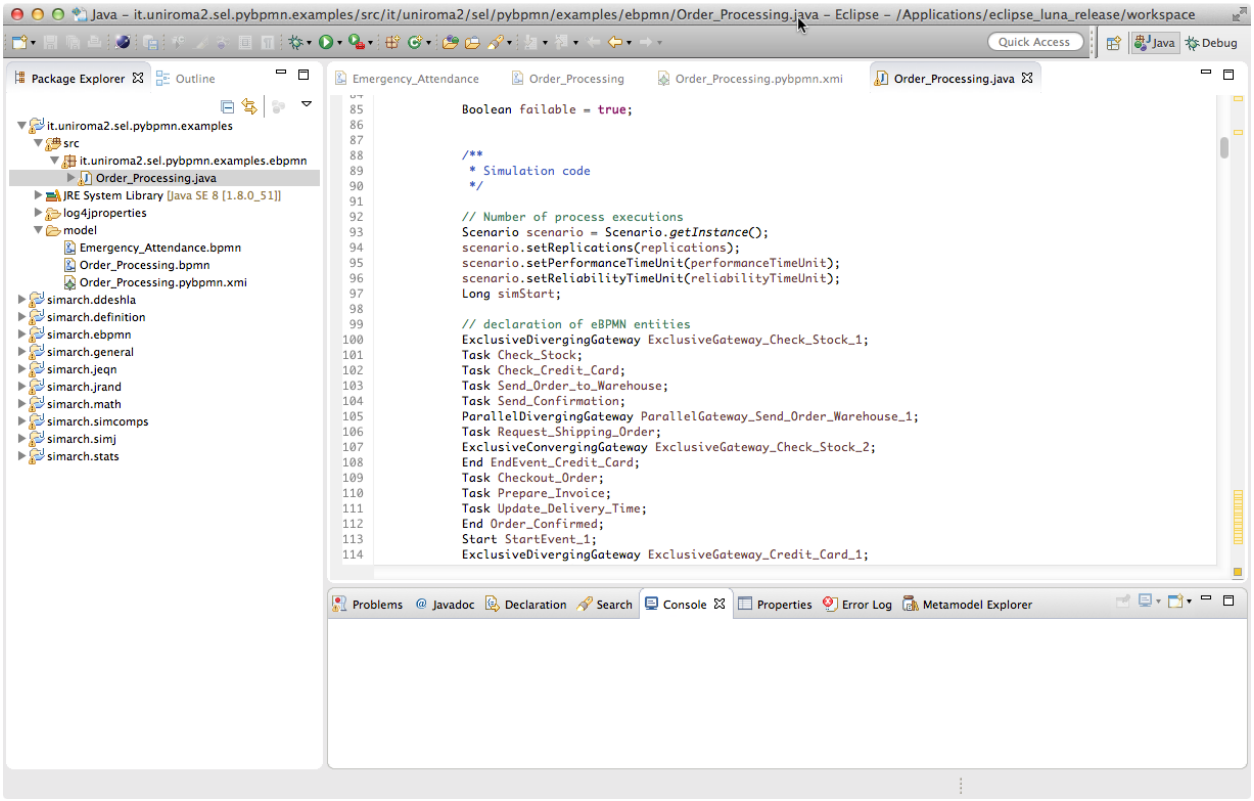

Figure 5: eBPMN simulation code generated by the model-to-text transformation.

4.5 Model-Driven Toolchain

The Eclipse Modeling Project tools and the developed plugins can be assembled together thus providing an integrated environment that can be used by the analyst to effectively design, simulate and evaluate the business process.

The enacting platform for the presented toolchain is the Eclipse Modeling Framework (EMF), which includes:

- the *BPMN2 Modeler*, to define BPMN models with textual annotations conforming to the PyBPMN syntax;
- the *contextual menu*, to execute the BPMN to PyBPMN and the PyBPMN to eBPMN transformations;
- the *eBPMN simulation engine*, to execute the simulation code and evaluate the business process behavior.

5 CASE STUDY

In this section, a simple case study is introduced in order to present the actual application of the proposed BPMN-based modeling and simulation approach.

In the case study, each of the steps presented in Section 4 are followed and described in detail.

5.1 Scenario Definition

The case study refers to an emergency attendance process inspired by the process used as running example in the Bizagi Process Modeler user guide (Bizagi 2019), which is here briefly summarized.

The emergency attendance process begins with the reception of an emergency report by the call center agent. The call center agent gathers all the information about the person affected by the emergency, along with the symptoms and the patient address. The call center agent transmits the report to a nurse who analyzes the report and classifies the emergency according to its severity:

- **Green code**: low severity, the patient can be easily treated;
- **Yellow code**: medium severity, the patient requires some special attention but can be treated at the place of emergency;
- **Red code**: high severity, the patient must be taken to an hospital.

The kind of emergency response depends on the severity of the classification:

- Green code: the triage is assisted by a quick attention vehicle (QAV);
- Yellow code: the triage is assisted by a basic ambulance (BA);
- Red code: the triage is assisted by a fully equipped ambulance.

For green and yellow codes, the process is completed with the arrival at the place of the emergency. For red codes, the fully equipped ambulance takes the patient to the nearest hospital. During the transfer a nurse prepares the paperwork that is used by the receptionist at the hospital to admit the patient.

5.2 Business Process Specification

At the first step, the business analyst specifies the BP as a BPMN model identifying the main activities and the participants in charge of their execution.

The business analyst identifies the logical relationships and order between activities, also specifying the following probabilities of each alternative flow (e.g., based on statistical analysis on emergency reports):

- Red codes: 50%;
- Yellow codes: 30%;
- Green codes: 20%.

5.3 Business Process Annotation

At the second step, the BPMN model is enriched with textual annotations specified by use of the PyBPMN syntax. For the sake of simplicity, this paper only addresses performance properties.

The business analyst identifies the quantities for the available resources, as shown in Table 1.

Then, the business analyst associates activities and resources, specifying the required amount of work for each activity, as reported in Table 2.

Finally, the business analyst specifies the aforementioned information in the BPMN model as text annotations that conform to the PyBPMN grammar.

The annotated BPMN model is depicted in Figure 6.

5.4 Model Transformations

The `BPMN_to_PyBPMN` model-to-model transformation takes as input the annotated BPMN model, as produced by the analyst, to automatically generate the corresponding PyBPMN model.

Such transformation creates, in the target model, the PyBPMN elements that correspond to the text annotations in the BPMN model.

Table 1: Available quantity for resources involved in the emergency attendance process.

Resource	Quantity
Call center agent	2
Nurse	3
Fully equipped ambulance	3
Quick attention vehicle	2
Basic ambulance	2
Hospital receptionist	2

Table 2: Processing times and resources for the activities performed in the emergency attendance process.

Activity	Processing time [min]	Resource
Receive emergency report	4	Call center agent
Classify triage	5	Nurse
Manage patient entry	11	Nurse
Pick up patient	20	Fully equipped ambulance
Arrive at patient place QAV	7	Quick attention vehicle
Arrive at patient place BA	10	Basic ambulance
Authorize entry	4	Hospital receptionist

According to the model-driven toolchain introduced in Section 4.5, the `PyBPMN_to_eBPMN` model-to-text transformation takes as input the obtained PyBPMN model to eventually generate the corresponding eBPMN simulation code.

5.5 Business Process Simulation

At the simulation step, the eBPMN code generated in the previous step is executed to predict the behavior of the BP.

The eBPMN simulation code is used to create a large number of process instances, execute these instances step-by-step and collect information on process measures.

The business analyst can use such measures to predict the process performance and check if the initial requirements are met by the considered process configuration in terms of activities and associated resources.

5.6 Business Process Evaluation

The analyst can also evaluate the process behavior under various workloads, in order to verify the BP behavior for different input conditions. The results of this what-if analysis are illustrated in Figures 7 and 8, which depict the cycle times and resources utilization for red codes, yellow codes and green codes, as well as the average process time, when the emergency report rate varies from 0.01 per minute (one report every 100 minutes on average) to 0.2875 per minute (about one report every 3.48 minutes on average).

Finally, if the requirements are not satisfied or the what-if analysis suggests some modifications to the process, the business analyst can evaluate alternative configurations by reusing the toolchain with a different amount of resources.

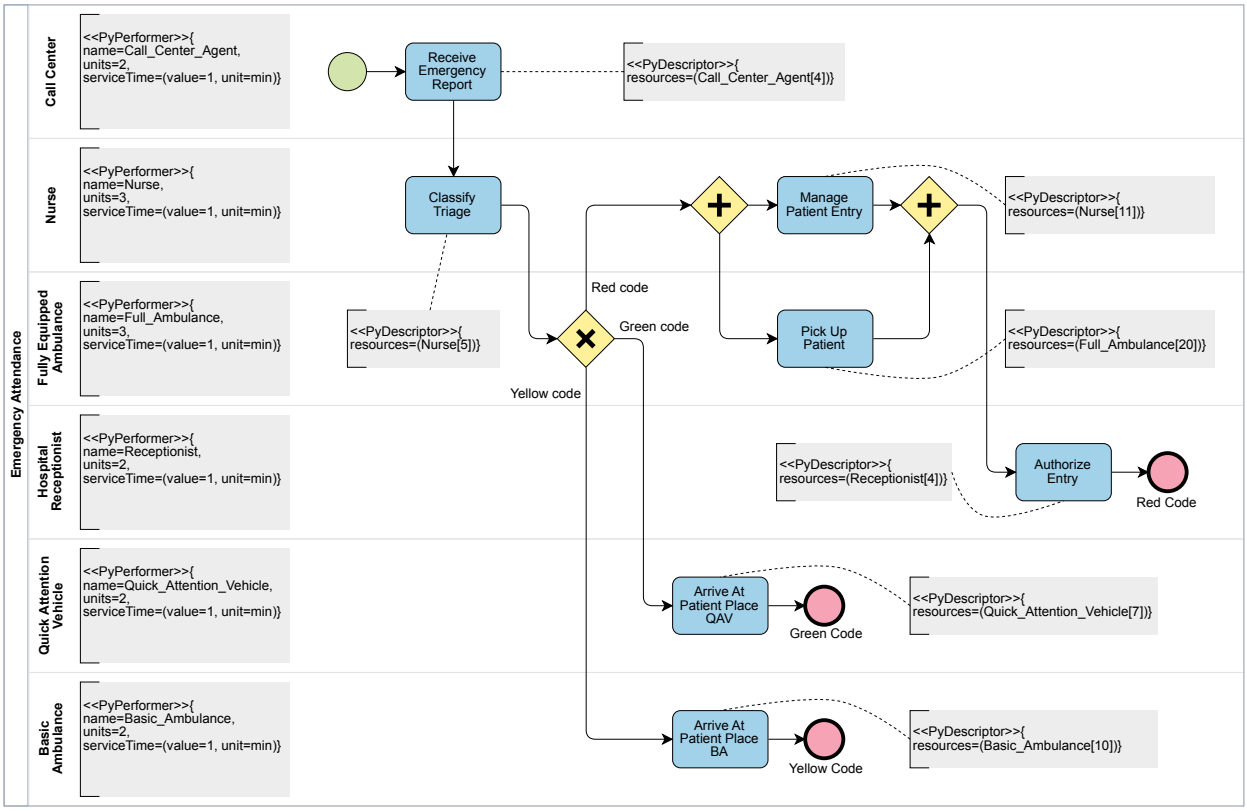

Figure 6: Annotated BPMN model of the emergency attendance process.

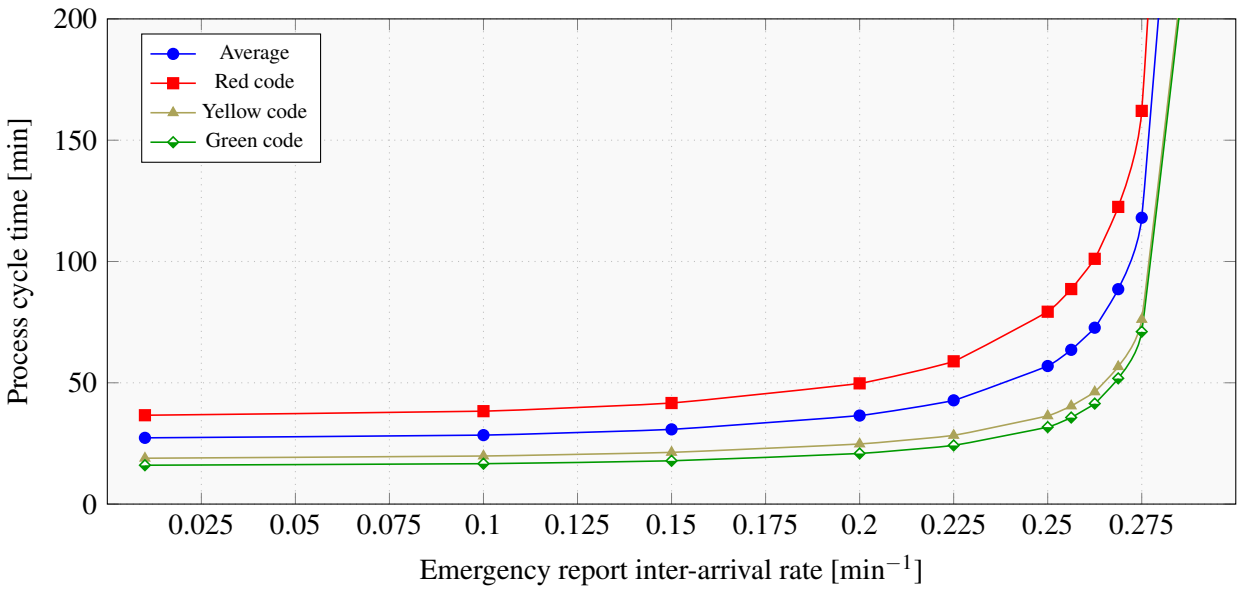

Figure 7: Cycle times for the emergency attendance process.

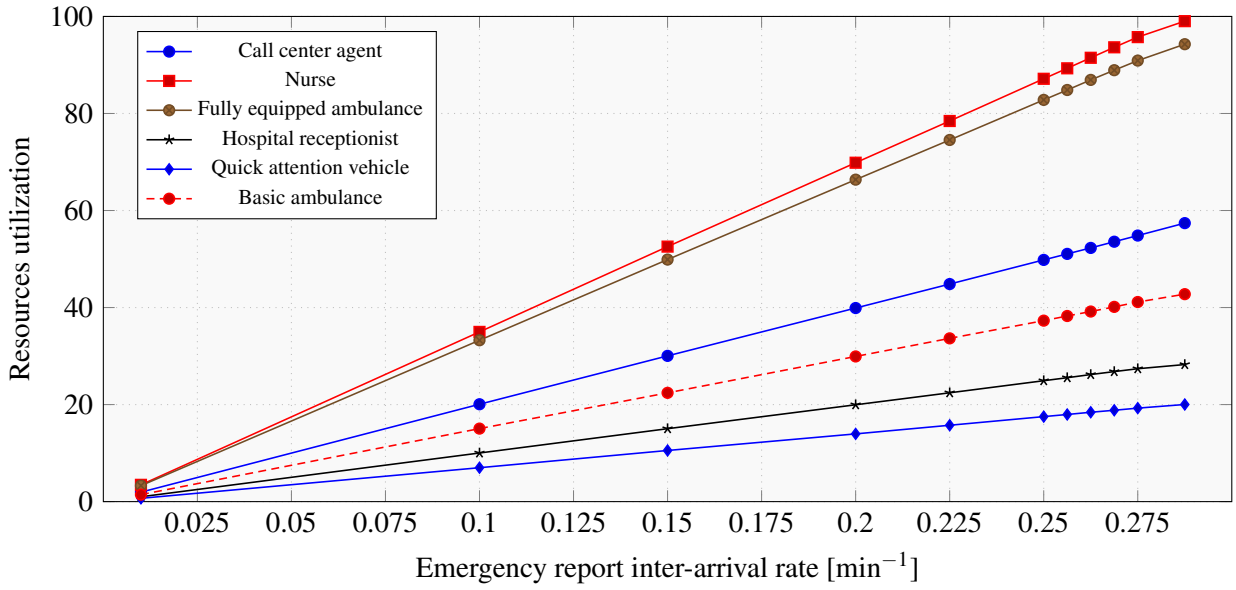

Figure 8: Resources utilization for the emergency attendance process.

6 CONCLUSIONS

This paper has presented a largely automated approach for BPMN-based business process simulation. In order to carry out the simulation-based analysis of a given business process, the proposed approach first annotates the BPMN model with what is necessary to make the model executable (i.e., performance parameters, execution resources, and expected workload), and then automatically maps the annotated model into simulation code ready to be executed.

In this respect, the paper has introduced the PyBPMN extension, for BPMN model annotation, and the eBPMN domain-specific language, for specifying and executing the simulation code.

The proposed approach allows business analysts to reduce the effort and the know-how typically required for using simulation-based analysis techniques, thanks to a lightweight extension mechanism and to the automation layer introduced by model transformations. A toolchain has been described that enables business analysts to easily and effectively design, simulate and evaluate their business processes.

The paper also includes an example application of the proposed approach to an emergency attendance process, so to concretely illustrate the various, mostly automated, steps that enact BPMN-based business process modeling and simulation.

REFERENCES

Bizagi 2019. *Welcome to the Bizagi Modeler and Modeler Services documentation*. http://help.bizagi.com/process-modeler/en, accessed 24th April.

Bocciarelli, P., and A. D'Ambrogio. 2011a. "A BPMN Extension for Modeling Non Functional Properties of Business Processes". In *Proceedings of the 2011 Spring Simulation Multi-Conference (Symposium on Theory of Modeling and Simulation, DEVS-TMS)*, 160–168. San Diego, CA, USA: Society for Computer Simulation International.

Bocciarelli, P., and A. D'Ambrogio. 2011b. "Performability-Oriented Description and Analysis of Business Processes". In *Business Process Modeling: Software Engineering, Analysis and Applications*, edited by J. A. Beckmann, Chapter 1, 1–36. Hauppauge, NY, USA: Nova Science Publisher.

Bocciarelli, P., and A. D'Ambrogio. 2014. "A Model-Driven Method for Enacting the Design-time QoS Analysis of Business Processes". *Software & Systems Modeling* 13(2):573–598.

Bocciarelli, P., A. D'Ambrogio, A. Giglio, E. Paglia, and D. Gianni. 2014. "A Transformation Approach to Enact the Design-Time Simulation of BPMN Models". In *2014 IEEE 23rd International WETICE Conference*, edited by S. M. Reddy, 199–204. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bocciarelli, P., A. D'Ambrogio, and E. Paglia. 2014. "A Language for Enabling Model-Driven Analysis of Business Processes". In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, edited by L. Ferreira Pires, S. Hammoudi, J. Filipe, and R. Csar das Neves, MODELSWARD'14. Setbal, Portugal: SCITEPRESS - Science and Technology Publications, Lda.

D'Ambrogio, A., E. Paglia, P. Bocciarelli, and A. Giglio. 2016. "Towards performance-oriented perfective evolution of BPMN models". In *6th International Workshop on Model-Driven Approaches for Simulation Engineering*, edited by F. Barros, X. Hu, H. Prahofer, and J. Denil, 15:1–15:8. San Diego, CA, USA: Society for Computer Simulation International.

Desel, J., and T. Erwin. 2000. "Modeling, Simulation and Analysis of Business Processes". In *Business Process Management, Models, Techniques, and Empirical Studies*, edited by W. M. P. van der Aalst, J. Desel, and A. Oberweis, 129–141. Berlin, Heidelberg, Germany: Springer-Verlag.

Dumas, M., M. La Rosa, J. Mendling, and H. A. Reijers. 2013. *Fundamentals of Business Process Management*. Berlin, Heidelberg, Germany: Springer-Verlag.

Eclipse Foundation 2019a. *BPMN2 Modeler*. https://www.eclipse.org/bpmn2-modeler, accessed 3rd May.

Eclipse Foundation 2019b. *Eclipse Modeling Framework (EMF)*. https://eclipse.org/modeling/emf, accessed 3rd May.

Gianni, D., A. D'Ambrogio, and G. Iazeolla. 2008. "A layered architecture for the model-driven development of distributed simulators". In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & workshops*. Brussels, Belgium: ICST.

Hook, G. 2011. "Business Process Modeling and Simulation". In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. Creasey, J. Himmelspach, K. White, and M. Fu, 773–778. Piscataway, New Jersey: IEEE.

Kamrani, F., R. Ayani, and A. Karimson. 2010. "Optimizing a Business Process Model by Using Simulation". In *IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 1–8. Atlanta, GA.

OASIS 2019. *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, accessed 17th September.

OMG 2019a. *Business Process Modeling Notation (BPMN), Version 2.0*. https://www.omg.org/spec/BPMN/2.0, accessed 3rd May.

OMG 2019b. *Meta Object Facility (MOF) 2.0 Query/View/Transformation*. https://www.omg.org/spec/QVT, accessed 17th September.

OMG 2019c. *MOF Model to Text Transformation Language*. https://www.omg.org/spec/MOFM2T, accessed 3rd May.

Scowen, R. S. 1993. "Extended BNF–a generic base standard". In *Proceedings of the 1993 Software Engineering Standards Symposium (SESS'93)*, 25–34. Los Alamitos, CA, USA: IEEE Computer Society.

van der Aalst, W. M. P. 2013. "Business Process Management: A Comprehensive Survey". *ISRN Software Engineering* 2013:1–37.

van der Aalst, W. M. P., J. Nakatumba, A. Rozinat, and N. Russell. 2010. "Business Process Simulation: How to get it right?". In *Handbook on Business Process Management, International Handbooks on Information Systems*, edited by J. vom Brocke and M. Rosemann, 317–342: Berlin, Heidelberg, Germany: Springer-Verlag.

Weske, M. 2012. *Business Process Management: Concepts, Languages, Architectures*. 2nd ed. Berlin, Heidelberg, Germany: Springer-Verlag.

AUTHOR BIOGRAPHIES

PAOLO BOCCIARELLI is a postdoc researcher at the University of Rome Tor Vergata (Italy). His research addresses the application of M&S and model-driven development to software and systems engineering and business process management. His email address is paolo.bocciarelli@uniroma2.it.

ANDREA D'AMBROGIO is Associate Professor at the Dept. of Enterprise Engineering of the University of Roma Tor Vergata (Italy). His research interests are in the fields of model-driven software engineering, dependability engineering, distributed and web-based simulation. His email address is dambro@uniroma2.it.

ANDREA GIGLIO is a postdoc researcher at the University of Rome Tor Vergata (Italy). His research interests include model-driven system and software engineering and business process management. His email address is andrea.giglio@uniroma2.it.

EMILIANO PAGLIA is a postdoc researcher at the University of Rome Tor Vergata (Italy). His research interests include model-driven engineering and business process modeling and analysis. His email address is emiliano.paglia@uniroma2.it.