A Language for Enabling Model-Driven Analysis of Business Processes

Paolo Bocciarelli, Andrea D'Ambrogio and Emiliano Paglia
Department of Enterprise Engineering, University of Rome Tor Vergata, Rome, Italy

Keywords: Business Process, BPMN, Model Transformation, Domain-specific Language, Simulation, Performance.

Abstract: The use of simulation-based approaches for the analysis of business processes enables the design-time prediction of the process behavior and/or the operation-time process reconfiguration. However, the effectiveness of BP simulation is still limited for several reasons (e.g., lack of simulation know-how of BP analysts, simulation model parameters that can be hard to gather, large semantic gap between the business process model and the simulation model). To overcome such limitations, this paper introduces a model-driven method to automatically build the executable simulation code of a business process from its abstract definition in BPMN, the standard language for specifying business processes. The simulation code is specified in eBPMN, a novel domain-specific language that has been designed and implemented according to the BPMN execution semantics.

1 INTRODUCTION

Modern enterprises are increasingly interested in analyzing and optimizing the *business processes* (*BPs*) that define the set of tasks executed to deliver services and/or produce goods. A BP is defined by the set of tasks that are orchestrated for delivering value to customers and commercial partners.

In this context, business analysts claim for methods and tools to specify business goals, analyze the model of a BP and predict at design time its behavior to assess whether or not the business goals can be achieved. It is thus essential to quantitatively analyze a specified BP to study the characteristics affecting its overall quality. The most relevant example of BP quantitative analysis is the performance analysis (van der Aalst et al., 2010).

In (Kamrani et al., 2010; Hook, 2011; van der Aalst et al., 2010) simulation has been identified as a key technique for BP performance analysis. However, the effectiveness of BP simulation is still limited, mainly for the following reasons:

- lack of simulation know-how of BP analysts;
- costs and difficulties in retrieving and analyzing the data required for simulation model parameterization;
- large semantic gap between the business process model and the simulation model;
- use of models that may be (partially) incorrect or may not be at the right level of abstraction.

In order to overcome such limitations, we have proposed a model-driven method for the generation of executable simulation code from BP models specified by use of the BPMN (Business Process Model and Notation) language (Bocciarelli and D'Ambrogio, 2012; Bocciarelli et al., 2012). The method has been used to generate queueing-based analysis models of the BP under study. The simulation-based analysis of such models, which are specified by use of the EQN (Extended Queueing Network) formalism, yields the design-time prediction of the BP behavior, to assess whether or not it meets the required constraints.

In this work we propose a significant extension of the previous contribution. Specifically, we introduce a model-driven method that exploits a novel language, named *eBPMN*, to automatically build BP executable simulation code from standard BPMN models. eBPMN is a domain-specific language that has been designed and implemented according to the BPMN execution semantics, thus obtaining a compact and precise executable model of the input BP model, without the need of introducing additional formalisms, such as queueing-based modeling languages. As a consequence, it can be effectively used by business analysts who do not have specific queueing theory skills.

The proposed method exploits both *PyBPMN* (Bocciarelli and D'Ambrogio, 2011b), a BPMN extension to specify performance-related properties of a BP, and *SimArch* (Gianni et al., 2011), a layered infrastructure to ease the specification and the

Bocciarelli P., D'Ambrogio A. and Paglia E..
A Language for Enabling Model-Driven Analysis of Business Processes.
DOI: 10.5220/0004712603250332
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 325-332
ISBN: 978-989-758-007-9

implementation of discrete event simulation models.

Finally, the proposed approach overcomes the limitations of the several commercially-available BPM tools that show a low degree of customizability (or result not customizable at all), being tied to the specific implementation technologies adopted for modeling and simulating (or often simply animating) the BP behavior.

The rest of the paper is structured as follows. Section 2 reviews relevant contributions dealing with BP analysis. Section 3 briefly provides an overview of the technical background at the basis of this work. Section 4 describes eBPMN, while Section 5 illustrates the proposed model-driven method that yields the executable eBPMN simulation code from the abstract BPMN model. Finally, Section 6 outlines the conclusions.

2 RELATED WORK

This section reviews the existing literature dealing with BP analysis. In this respect, as stated in Section 1, this work extends and improves our previous contributions (Bocciarelli and D'Ambrogio, 2012; Bocciarelli et al., 2012), which introduce a model-driven method for the generation of executable queueing-based simulation code from BPMN models. Similarly to such contributions, this work exploits model-driven techniques and the SimArch architecture to ease the automated generation of executable simulation code of the BP under study. Differently, this work introduces eBPMN, a novel simulation language built on top of SimArch and based on the BPMN 2.0 execution semantics.

Approaches that deal with the use of executable models can be found, e.g., in (Tatibouet et al., 2013; Weyprecht and Rose, 2011). The use of *fUML* in simulation processes is discussed in (Tatibouet et al., 2013), which proposes new mechanisms for delegating the execution control, in order to provide a basic support for discrete-event simulation. The limitations of the reference implementation of fUML are discussed in (Weyprecht and Rose, 2011), which states that fUML does not include the concept of time and does not provide mechanisms to observe elements during the simulation. A new implementation of fUML is then synthetically discussed, which aims to provide all the features required to effectively use fUML for simulation purposes. Such aforementioned approaches are founded on the adoption of the fUML notation. In this respect, it should be noted how fUML could be easily reused in the BPM context, according to the following considerations: i) (White, 2004) states that fUML activity diagrams and BPMN models can be considered semantically equivalent; ii) the abstract BP model can be specified as a fUML activity diagram either directly or transforming a BPMN model by use of existing model-driven approaches, as discussed in (Bocciarelli and D'Ambrogio, 2012). Nevertheless, as underlined in (Tatibouet et al., 2013; Weyprecht and Rose, 2011), fUML shows several limitations and the reference implementation needs to be extended in order to be used in real-world simulation cases. Differently from contributions that make use of fUML, this work is founded on the use of a domain-specific simulation language, built according to the BPMN 2.0 specification.

The optimization of BPs by use of performance-based simulation or analytical techniques can be found in (Kamrani et al., 2010; Yang et al., 2010; Grefen et al., 2000). A BPMN extension to enable the association of performance information to task and activity constructs is introduced in (Kamrani et al., 2010). The proposed approach exploits such extension to estimate an overall measure of the BP performance by simulating a model of human agents performance. The performance metrics are finally used to optimize the allocation of tasks to agents. The application of discrete event simulation to workflows is discussed in (Yang et al., 2010). The paper proposes a framework for process simulation based on multi-agent systems to support flexible activity scheduling. Differently from the aforementioned contributions, this work makes use of the eBPMN language for implementing and executing the corresponding simulation. Moreover, the proposed approach concretely exploits SimArch and model-driven core standards, such as MOF, XMI and QVT, to obtain a significant reduction of the effort needed for implementing the relevant simulation.

3 BACKGROUND

In order to provide a clear understanding of the technical basis of this paper work, the following subsections briefly outline both the PyBPMN extension and the SimArch architecture.

3.1 Performability-enabled BPMN

BPMN is a standard notation for the high-level representation of business processes and is typically used at the early stages of the business process lifecycle. Despite its pervasiveness and completeness, BPMN does not support the characterization of the

BP in terms of non functional properties, such as performance or reliability (Saeedi et al., 2010). Currently, BPMN descriptions cannot be used to specify overall performance constraints (e.g., the response time associated to the entire BP execution) or task properties (e.g., execution time of a single process task). To overcome such limitations we have introduced *PyBPMN* (*Performability-enabled BPMN*), a lightweight BPMN extension that addresses the specification of performance and reliability properties of a BP (Bocciarelli and D'Ambrogio, 2011b; Bocciarelli and D'Ambrogio, 2011a).

More specifically, PyBPMN is obtained as a BPMN metamodel extension that provides annotations to specify workloads, performance properties and reliability properties. In this work PyBPMN is specifically used to drive the generation of performance-oriented simulation models of BPs, by use of the following metaclasses:

- *ClosedPattern:* to represent the workload in terms of a fixed number of active or potential users/jobs.
- *OpenPattern:* to represent the workload in terms of a stream of requests that arrive at a given rate in some predetermined pattern.
- *PaService:* to represent the execution demand of tasks. It is specified by a set of attributes, such as *unit*, which describes the unit of measure, and *value*, which specifies the service time value.

A detailed description of PyBPMN can be found in (Bocciarelli and D'Ambrogio, 2013).

3.2 SimArch

SimArch is a layered software architecture that provides a general-purpose and event-based simulation infrastructure upon which domain-specific simulation components can be implemented and seamlessly used in local or distributed execution environments (Gianni et al., 2011).

The architecture, depicted in Figure 1, consists of five layers: distributed computing infrastructure (Layer 0), distributed discrete event simulation services (Layer 1), discrete event simulation services (Layer 2), simulation components (Layer 3) and simulation model (Layer 4). SimArch users specify simulation models at Layer 4 by use of a domain-specific simulation language. Currently, SimArch is provided with jEQN, a Java-based simulation library that implements a domain-specific language for the specification of EQN models.

Layer 0 is the distributed computing infrastructure, which can be either a simulation-specific infrastructure, such as the High Level Architecture

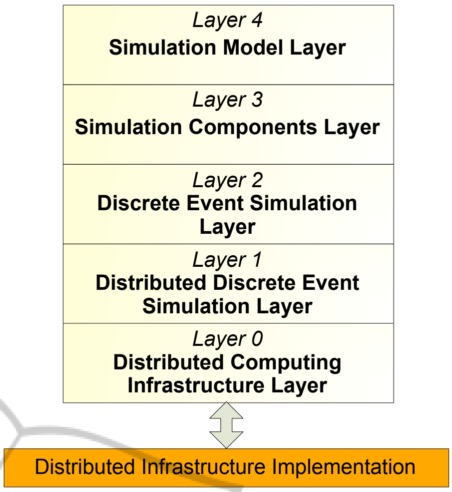

Figure 1: Simarch layered architecture.

(HLA), or a general-purpose infrastructure, such as a cloud-based platform. Layer 1 implements an abstraction of a distributed discrete event simulation and contributes to make the specific distributed infrastructure transparent to the uppermost layer. Layer 1 provides typical distributed simulation services such as synchronization or event dispatching, used by Layer 2 to provide a transparent discrete event simulation abstraction on top of local or distributed execution environments.

Layers from 0 to 2 constitute the backbone of the SimArch architecture and provide services to build discrete event simulations that can be transparently executed either in local or distributed execution contexts. Upper levels are instead committed to specify and implement domain-specific simulation systems. Specifically, Layer 3 contains the implementation of a domain-specific simulation language, which is used ad Layer 4 to define specific simulation models.

Currently, the Layer 2 implementation supports the process interaction simulation paradigm and is founded on the following concepts:

- *simulation engine*, which is the engine that executes the simulation system;
- *simulation entity*, which represents a building block used to build simulation systems;
- *port*, which belongs to a simulation entity and constitutes its interaction point; ports allow the exchange of events among entities;
- *link*, which represents the logical connection among entity ports;
- *event*, which represents something that occurs at a particular instant in time and affects the state of the system.

Interested readers are referred to the SimArch website (`https://sites.google.com/site/simulationarchitecture`) for additional details.

4 eBPMN

eBPMN is a domain specific simulation language built according to the BPMN 2.0 execution semantics (OMG, 2011) and thus close to the domain experts knowledge. Similarly to jEQN, eBPMN is built on top of SimArch and its implementation is composed of a set of Java-based primitives. Each primitive implements the execution semantics of a given BPMN construct. With regards to the SimArch layered architecture (see Figure 1), eBPMN is positioned at Layer 3 and uses/extends the Layer 2 services, classes and interfaces to enable the specification of BP simulation models at Layer 4. An architectural view of the eBPMN implementation is summarized in the class diagram depicted in Figure 2, where classes provided by the SimArch reference implementation are shown in gray background color.

According to the SimArch specification, a Layer 3 implementation is centered around the `ComponentLevelEntity` class that implements the `Layer2toLayer3Interface` and provides the simulation and synchronization services to manage events and to control the component execution. Table 1 summarizes a description of the eBPMN primitives that implement a subset of the elements provided by the BPMN 2.0 notation. As specifically described in Subsection 4.1, this work is a first step toward a full coverage of the BPMN 2.0 specification.

4.1 Implementation Assumptions

In order to simplify the space of the problem and, ultimately, to limit the implementation of the first release to the core BPMN elements, the currently available eBPMN version is built on top of the following assumptions:

- interactions among several participants must be expressed in terms of a *collaboration*. The use of a `choreography` model is not currently supported;
- the use of both multiple incoming and outgoing *sequence flows* is not permitted. Converging and diverging flows must be explicitly represented by use of *gateway* elements;
- each *Pool* must include either a *start event* element or a *receive task* element that activates the related process execution;
- conditions associated on outgoing *sequence flows* of *inclusive* and *exclusive* gateways must be expressed in terms of either the branching *probability* or the number of *iterations* to be performed;
- *Event-based gateways* are not currently supported.

5 MODEL-DRIVEN METHOD FOR BP SIMULATION

Figure 3 illustrates the method that exploits eBPMN to automate the simulation-based analysis of BPs. The proposed method has been built upon our previous work (Bocciarelli and D'Ambrogio, 2011b; Bocciarelli and D'Ambrogio, 2012), which has been revised and extended to achieve the following objectives:

- **Use of eBPMN:** methods proposed in previous contributions adopt a two steps approach that first generates an intermediate and abstract model (i.e., a UML Activity Diagram) and then uses such a model to generate a performance model based on queueing-based formalisms, such as EQN. Differently, this work integrates and exploits the eBPMN language and does not require any UML intermediate model. Moreover, being sternly linked to the BPMN 2.0 specification, eBPMN contributes to provide a more accurate simulation of the BP behavior.
- **Use of Visual Tools:** in order to allow the specification of the non-functional properties of BPs and to drive the automated execution of the simulation-based analysis, methods proposed in our past works are founded on the lightweight PyBPMN extension. One of the most relevant limitation of such an approach is the unavailability of visual tools to represents PyBPMN models. In this respect, this work introduces a novel model transformation that allows BP analysts to specify the BPMN model by use of BPMN-compliant visual tools and effortlessly annotate the model by use of the PyBPMN extension.

With regards to the use of visual tools, it should be underlined that alternative solutions based on the BPMN native extension mechanism (OMG, 2011) have been evaluated. Even though such an approach is promoted as the standard BPMN extension mechanism, it is affected by the following limitations (Stroppi et al., 2011):

- it does not define a methodological approach to design the extension;

Table 1: Description of the eBPMN primitives.

Class Name	Description
`FlowObject`	The building block of a BPMN model.
`Activity`	The execution of an `Activity` object starts as a token incomes. To model the execution of a process step, an `Activity` object holds the incoming token for the time required to execute the related process step and send it through the outgoing sequence flow. The `Activity` class constructor is parameterized according to the performance annotations specified in the PyBPMN model.
`ServiceTask`	An automated step that is executed by invoking an external web service. As the PyBPMN annotations are used to specify web service request and response messages, the behavior of the `ServiceTask` object takes into consideration the time required to send, receive and elaborate such messages.
`SendTask`	Upon activation, a `SendTask` object sends a message through the outgoing `MessageFlow` that causes the notification of an `IncomingMessage` event at the recipient side, then it terminates its execution.
`ReceiveTask`	A `ReceiveTask` class can be used either as an intermediate task or as an initial node. In the former case the `ReceiveTask` creates a new `Token` object and routes it through the outgoing `SequenceFlow`. In the latter case it waits until an `IncomingMessage` event occurs.
`StartEvent`	A source for a process execution and instantiation of a new `Token` object within the process workflow. The `StartEvent` class constructor is parameterized according to the workload annotations specified in the PyBPMN model.
`EndEvent`	A sink for all incoming tokens.
`Gateway`	Probability-based and iteration-based branching conditions.
`ParallelGateway`	Synchronization of multiple concurrent flows (i.e., *join* behavior) and activation of new concurrent threads on outgoing sequence flows (i.e., *fork* behavior).
`ExclusiveGateway`	Upon activation, the `ExclusiveGateway` class routes the incoming token to one of the associated outgoing `SequenceFlow` elements, according to the related condition that must be expressed in terms of either the branching *probability* or the number of *iterations* that have to be carried out.
`InclusiveGateway`	Upon execution, the incoming tokens are consumed and some tokens are generated and routed to a subset of the outgoing sequence flows. The branching condition must be expressed in terms of either the branching *probability* or the number of *iterations* that have to be carried out.
`Pool`	A thread of simulation execution that groups the elements associated to each participant and allows to collect process-related statistic data.
`SequenceFlow`	Connection link between two `ObjectFlow` classes, in order to allow the routing of tokens within the simulation model.
`MessageFlow`	Connection link between a `SendTask` class and a `ReceiveTask` class, in order to route `Message` elements
`Data`	Interface that specifies the payload of exchanged events.
`Message`	Data exchanged by a pair of `SendTask` and `ReceiveTask` elements
`Token`	The `Token` class manages the activities execution flow. It owns two attributes: `currentTime`, which stores the current value of the *simulation time*, and `bornTime`, which holds the *simulation time* value at the moment of the object instantiation.
`EventType`	The handling of events in eBPMN is fully based on the SimArch reference implementation. The `EventType` enumeration implements the two types of events that have to be managed by the underlying DES engine for the BP simulation purposes (i.e., *Incoming token* and *Incoming message*).

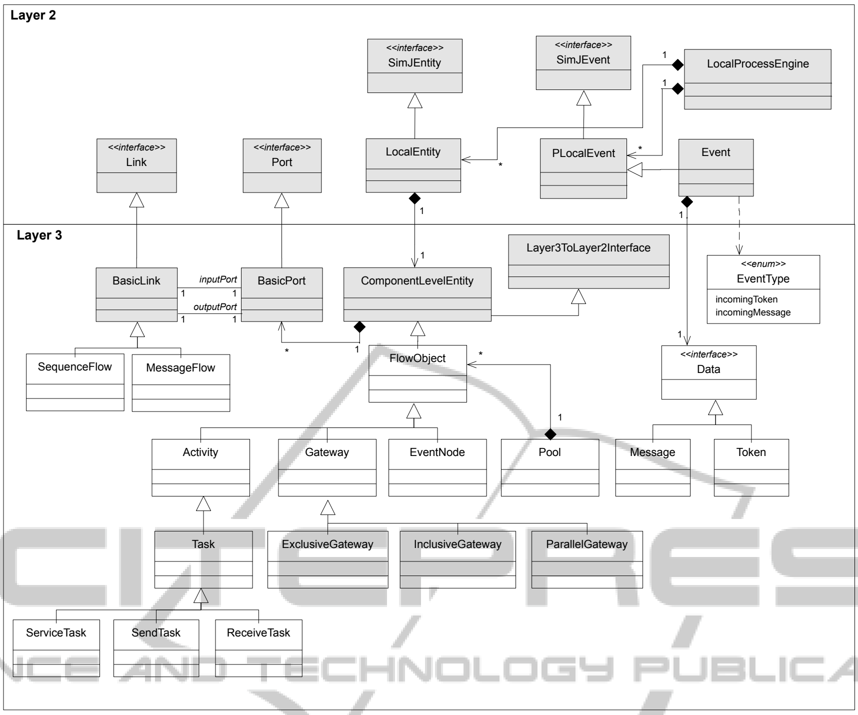

Figure 2: eBPMN architecture.

- the BPMN specification does not provide any visual notation to represent the extension.

As a consequence, the concrete support of the extension mechanism by open source and commercial tools is still limited in practice. The Eclipse Foundation provides a well documented method to build a plugin, based on the BPMN Modeler (Eclipse Foundation, 2013), whose implementation is based on the BPMN native extension mechanism. In such an approach, the BPMN extension is provided as an *Ecore* metamodel that is in turn used to create the *Java* code to define novel BPMN `flow object`. Differently, the proposed method (and specifically the PyBPMN extension) implementation leverages model-driven standards and tools so that it can be easily customized to fit specific needs.

According to Figure 3, at the first step, the business analyst gathers both functional and non-functional requirements of the BP under study and specifies the relevant BPMN model. While the functional requirements are used to define the flow of tasks and activities that constitute the BPMN model, the non-functional requirements are included as comments associated to the related `Activity` and `Process` objects. Such comments have to be specified according to the following syntax:

```
MetaclassName:AttributeName={tag1=value1,
tag2=value2,...}
```

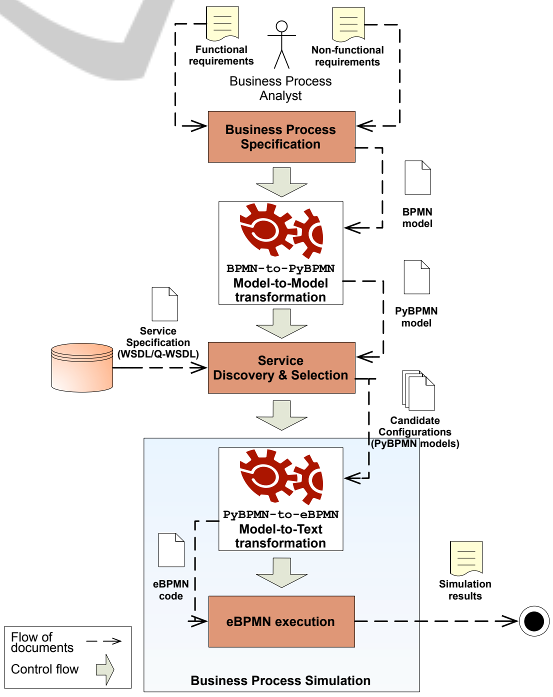

Figure 3: Model-driven BP simulation method.

where metaclasses and attributes are specified according to the PyBPMN extension and tags are parameters that quantify the attribute according to given datatypes, as specified in (Bocciarelli and D'Ambrogio, 2013).

As an example, a task that requires *200 ms* to be executed is specified by attaching the following comment to the corresponding `Task` element.

```
PaQualification:serviceTime = {value=200,
unit='ms', source='est', dir='decr',
statQ='max'}
```

where the tags *value*, *unit*, *source*, *dir* and *statQ* give the value, the unit of measure, the origin, the type of order relation used for values comparison and the type of statistical value of the attribute, respectively.

Then the `BPMN-to-PyBPMN` *model-to-model* transformation is executed in order to derive the PyBPMN model that corresponds to the BPMN input model.

A service discovery may also take place to retrieve the descriptions of concrete web services[1] that match the abstract service interfaces specified in the PyBPMN model.

The method proceeds with the simulation of the BP, which is carried out by first executing the `PyBPMN-to-eBPMN` *model-to-text* transformation, which takes as input the PyBPMN model and yields as output the eBPMN simulation code, and then executing the eBPMN code on top of the SimArch simulation engine.

The simulation results are finally used by the business analyst to evaluate design trade-offs or what-if scenarios and also to assess at *design time* the satisfiability of non-functional requirements.

5.1 Model Transformations

As stated in the previous section, the proposed method makes use of the `BPMN-to-PyBPMN` *model-to-model* transformation and the `PyBPMN-to-eBPMN` *model-to-text* transformation.

The `BPMN-to-PyBPMN` transformation has been implemented by use of the standard QVT-Operational (QVT-O) language (OMG, 2008) and it is carried out to transform a BPMN model into a PyBPMN model.

Specifically, the transformation maps the *annotation* elements that describe the non-functional properties of the BP under study (e.g., task service time and process workload characterization) to the corresponding PyBPMN elements (e.g., `PaService`, `OpenPattern` and `ClosedPattern` elements).

[1]The proposed method assumes that the tasks composing a BP are executed by automated resources implemented as web services and is fully compliant with both the standard WSDL description language and the performance-oriented Q-WSDL description (D'Ambrogio, 2006).

As an example, let us consider the mapping of a performance-related annotation. The QVT-O transformation engine parses the BPMN model (in XMI format) looking for well-formed `TextAnnotation` elements to be mapped to `PaService` elements in the target PyBPMN model. Then the transformation retrieves, from the input model, the `Association` element that links the `TextAnnotation` to the related `Activity` element, in order to associate the `PaService` element to the proper `Activity` element in the target model. Finally, the `TextAnnotation` is parsed to instantiate the `serviceTime` attribute in the target model and to assign a value to the related tagged values.

The `PyBPMN-to-eBPMN` transformation is a model-to-text transformation specified by use of the OMG's Model to Text (M2T) language and implemented on top of the Eclipse platform. The M2T language makes use of a template-based approach, in which placeholders for data to be extracted from models are used. These placeholders are expressions specified over metamodel entities.

As an example, the logic to instantiate a `Task` object is as follows:

```
[template public generateTask(t : Task)]
Task [t.id] = new Task();
[t.id].setName([t.name]);
// Other setters
[/template]
```

During the transformation the template `generateTask` (delimited by `template` and `/template` tags) is invoked for each `t` element of type `Task` that is in the source PyBPMN model. In order to avoid the creation of more than one object with the same name, the `id` attribute, which is unique throughout the model, is used as a reference to the object. The `name` attribute (which is not required to be unique) is used to set the name of the object through a setter method, as well as for the remaining attributes of the eBPMN object.

6 CONCLUSIONS

This paper has introduced a model-driven method for automating the transformation of BPMN models into simulation code specified by use of eBPMN, a domain-specific language for BP simulation.

Specifically, the proposed method first executes a model-to-model transformation, which takes as input the BPMN model of the BP under study and yields as output the corresponding PyBPMN model, and then executes a model-to-text transformation, which takes as input the PyBPMN model and yields as output the eBPMN executable simulation code, which is finally executed on top of the simulation engine provided by the underlying SimArch infrastructure.

The current eBPMN implementation includes a subset of the complete BPMN notation and represents a first step towards the implementation of a full-featured BP simulation-based analysis environment. Ongoing work includes both the implementation of additional BPMN constructs and an extensive experimentation and validation campaign on real-world cases.

REFERENCES

Bocciarelli, P. and D'Ambrogio, A. (2011a). A BPMN Extension for Modeling Non Functional Properties of Business Processes. In *Proceedings of the Symposium on Theory of Modeling and Simulation*, DEVS-TMS '11.

Bocciarelli, P. and D'Ambrogio, A. (2011b). Performability-oriented Description and Analysis of Business Processes. In Beckmann, J. A., editor, *Business Process Modeling: Software Engineering, Analysis and Applications*. Nova Publishers.

Bocciarelli, P. and D'Ambrogio, A. (2012). Automated performance analysis of business processes. In *Proceedings of the Symposium On Theory of Modeling and Simulation*, DEVS-TMS '12.

Bocciarelli, P. and D'Ambrogio, A. (2013). A Model-driven Method for Enacting the Design-time QoS Analysis of Business Processes. *Software & Systems Modeling*.

Bocciarelli, P., Pieroni, A., Gianni, D., and D'Ambrogio, A. (2012). A Model-driven Method for Building Distributed Simulation Systems from Business Process Models. In Rose, O. and Uhrmacher, A. M., editors, *Winter Simulation Conference*, page 227. WSC.

D'Ambrogio, A. (2006). A model-driven wsdl extension for describing the qos of web services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, pages 789–796, Washington, DC, USA. IEEE Computer Society.

Eclipse Foundation (2013). BPMN2 modeler. Website: http://www.eclipse.org/bpmn2-modeler/ documentation.php.

Gianni, D., D'Ambrogio, A., and Iazeolla, G. (2011). A software architecture to ease the development of distributed simulation systems. *Simulation*, 87(9):819–836.

Grefen, P., Aberer, K., Hoffner, Y., and Ludwig, H. (2000). Crossflow: Cross-organizational workflow management in dynamic virtual enterprises. *Computer Systems Science & Engineering*, 1(5):277–290.

Hook, G. (2011). Business process modeling and simulation. In Jain, S., Jr., R. R. C., Himmelspach, J., White, K. P., and Fu, M. C., editors, *Winter Simulation Conference*, pages 773–778. WSC.

Kamrani, F., Ayani, R., and Karimson, A. (2010). Optimizing a business process model by using simulation. In *IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*, pages 1–8, Atlanta, GA.

OMG (2008). Meta object facility (mof) 2.0 query/view/-transformation, version 1.0.

OMG (2011). *Business Process Modeling Notation (BPMN), version 2.0, http://www.omg.org/spec/BPMN/2.0/*.

Saeedi, K., Zhao, L., and Sampaio, P. R. F. (2010). Extending bpmn for supporting customer-facing service quality requirements. *Web Services, IEEE International Conference on*, 0:616–623.

Stroppi, L. J. R., Chiotti, O., and Villarreal, P. D. (2011). Extending bpmn 2.0: Method and tool support. In Dijkman, R. M., Hofstetter, J., and Koehler, J., editors, *BPMN*, volume 95 of *Lecture Notes in Business Information Processing*, pages 59–73. Springer.

Tatibouet, J., Cuccuru, A., Gérard, S., and Terrier, F. (2013). Principles for the realization of an open simulation framework based on fuml (wip). In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*, DEVS 13, pages 4:1–4:6, San Diego, CA, USA. Society for Computer Simulation International.

van der Aalst, W., Nakatumba, J., Rozinat, A., and Russell, N. (2010). Business Process Simulation: How to get it right? In *Handbook on Business Process Management, International Handbooks on Information Systems*, pages 317–342. Springer-Verlag.

Weyprecht, P. and Rose, O. (2011). Model-driven development of simulation solution based on sysml starting with the simulation core. In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pages 189–192, San Diego, CA, USA. Society for Computer Simulation International.

White, S. A. (2004). Process Modeling Notations and workflow patterns. *Workflow Handbook*, pages 265-294.

Yang, F., Shen, W., Tan, W., and Ghenniwa, H. (2010). A framework for service enterprise workflow simulation based on multi-agent cooperation. In *Proc. of the IEEE Int. Conference on Systems, Man and Cybernetics, Istanbul, Turkey, 10-13 October 2010*, pages 2587–2594. IEEE.