Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

RESOURCE MODELING IN BUSINESS PROCESS SIMULATION

Paolo Bocciarelli
Andrea D'Ambrogio

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

Gerd Wagner

Department of Informatics
Brandenburg University of Technology
Konrad-Wachsmann-Allee 5
Cottbus, 03046, GERMANY

ABSTRACT

Business Process (BP) models address the specification of the flow of events and activities, along with the dependencies of activities on resources. BP models are often analyzed by using simulation-based approaches. This paper focuses on resource modeling for BP modeling and simulation, by first introducing the most important concepts and discussing how resources are modeled in the standard BP modeling notation (i.e., BPMN) and in the area of Discrete Event Simulation. Then, the paper presents two newer BP modeling and simulation approaches, namely the Object Event Modeling and Simulation with the Discrete Event Process Modeling Notation (DPMN) based on the JavaScript-based simulation framework OESjs, and Performability-enabled BPMN (PyBPMN), with the Java-based simulation framework eBPMN. A simple but effective case study dealing with a pizza service process is used to illustrate the main features of the presented approaches.

1 INTRODUCTION

Business processes have been modeled (and simulated) both in the area of Discrete Event Simulation (DES) and in the area of Business Process Management (BPM) for a long time. However, research in both areas has not managed to develop a unified conceptual framework for business process (BP) modeling and simulation. In particular, in BPM, a standard BP modeling language, the BP modeling notation BPMN (officially called 'Business Process Model and Notation') has been established in 2005 (Ojbect Management Group 2014), while in DES no common modeling language was established and BPMN has been largely ignored.

Today, both areas still use different approaches to BP modeling and simulation. While DES tools, such as Arena, ExtendSim, Simio and AnyLogic, use their own proprietary modeling concepts/terminologies and diagram languages based on the Event Scheduling and Processing Network paradigms, BPMN simulation tools are not based on Event Scheduling, but on a peculiar Petri-Net-style token flow semantics. As a consequence of this schism, some DES tool vendors (Simul8 and Lanner L-Sim) offer a BPMN simulation tool in addition to their DES tool.

Business process (BP) models focus on describing the possible sequences of events and activities, based on conditional and parallel branching, but they also describe the dependencies of activities on resource objects, either *declaratively*, as in BPMN, by defining resource roles for activities, or *procedurally*, as in DES tools, by preceding and succeeding resource allocation and de-allocation steps.

In this paper we focus on resource modeling as an essential part of BP modeling and simulation. We elaborate the most important concepts for resource modeling and discuss how resources are modeled in DES and BPMN. Then we present two newer BP modeling and simulation approaches:

1. *Object Event Modeling and Simulation* with the *Discrete Event Process Modeling Notation (DPMN)* and the JavaScript-based simulation framework *OESjs*, and
2. *Performability-enabled BPMN (PyBPMN)* with the Java-based simulation framework *eBPMN*.

2 RESOURCE MODELING CONCEPTS

This section is based on Section 10.5 of (Wagner 2022).

For executing a *planned activity* (or *task*), certain resources are required and have to be allocated first. We can distinguish between 'active' (or *performer*) resources, such as human resources, robots or IT systems, which execute activities, and 'passive' resources, such as rooms or devices, which are used by performers for executing activities.

In the field of Business Process Management (BPM), there is a tendency to consider workflow processes as paradigmatic for business processes. However, workflow processes typically only have (human) performer resources, while business processes in general often have one or many passive resources in addition to a performer. Due to this tendency, BPM research often adopts a simplistic view of business processes.

E.g., in (Pufahl et al. 2021), resource allocation is defined as the "assignment of a process task to the most appropriate resource", while in general, it's the other way around: resources are assigned to a task. The authors also point out that the vast majority of BPM research studies on resource allocation make the assumption that a task requires just one resource and a resource cannot be used in more than one task at the same time. As opposed to this simplistic view of business processes in BPM, in DES, tasks can have more than one required resource, and resources may be used in more than one task at the same time.

It is important to distinguish between the passive resources required for performing an activity and the inputs required for performing a processing (or transformation) activity. While the passive resources used in an activity survive the activity, and can be reused in the next one, inputs are either processed, or transformed, into outputs of the activity, and cannot be reused for the next activity. In the literature, this conceptual distinction is often not made. Instead, e.g., in (Russell et al. 2004; Goel and Lin 2021), processing activity inputs are called 'consumable resources'.

Resources cannot be 'consumed' by activities, but they can be worn out by activities, such that if their degree of wear falls below a certain threshold, they are depleted and can no longer be used.

Since activities require resources, and resources are allocated to planned activities, it is crucial to understand the relationship between activities and resources. An important part of its meaning are *resource cardinality constraints* and *multitasking constraints*, as explained in the following subsection.

2.1 The Relationship between Activities and Resources

Business process models do not only describe the possible sequences of events and activities, but they also describe the dependency of activities on resources. For instance, in a BPMN process diagram, a Lane *L* may represent a resource role such that the diagram specifies, for any Activity rectangle *A* in *L*, that an activity of type *A* is performed by a resource from the pool that is implicitly associated to *L*. While using a container shape, such as a BPMN Lane, for assigning elements, such as activities, to the element represented by the container shape (viz, a resource role) is a good way of visually expressing an association, it does not allow expressing one-to-many or many-to-many associations between activities and resource roles.

Therefore, the relationship between activities and resources can be best described by an association in an information model, such as a UML class diagram. Figure 1 shows a one-to-one association between the activity type *examinations* and the object type *doctors* associating a doctor as a performer resource to an activity.

The association end at the side of the object type *doctors* is categorized as a *resource role* with the help of UML's stereotype notation. Such a binary association has a *multiplicity* expression at each end.

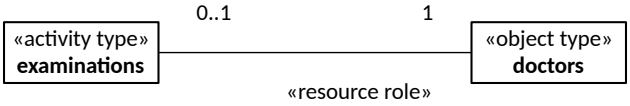

Figure 1: An examination requires a doctor as a (performer) resource.

The multiplicity at the *doctors* end is "1", which means that exactly one doctor is required for performing an examination, defining a *resource cardinality constraint*.

The association end at the side of the activity type *examinations* has the multiplicity "0..1", which means that, at any point in time, a doctor can be associated with, or perform, either no examination or at most one examination. Such a *multitasking constraint* defines if a resource is exclusively allocated to an activity, or to how many activities a resource can be allocated at the same time.

In an Object-Oriented (OO) programming approach, the computational meaning of a resource role is provided by a corresponding property in the class implementing the activity type. In the above example, the resource role association of *examinations* with *doctors* leads to a property *doctor* in the class *Examination*, as shown in Figure 2.

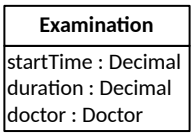

Figure 2: The class *Examination* implementing the corresponding activity type.

While the model shown in Figure 1 is an example of a one-to-one relationship between activities and resources, we consider the cases of one-to-many, many-to-one and many-to-many relationships in the following subsections.

2.1.1 One-to-Many Relationships

An example of a one-to-many relationship between activities and resources is the association between "load truck" activities and "wheel loader" resources shown in Figure 3.

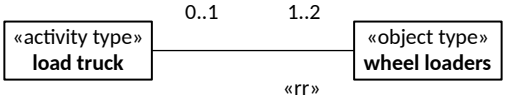

Figure 3: The loading of a truck requires at least one, and can be handled by at most two, wheel loaders.

Notice that in this diagram, the "resource role" designation of the association end at the side of "wheel loaders" has been abbreviated by the keyword "rr". Its multiplicity of 1..2 specifies a *resource minimum cardinality constraint* of "at least one" and a *resource maximum cardinality constraint* of "at most two", meaning that one wheel loader is required and a second one is admissible.

When an activity does not require, but admits of using more than one resource (of the same type), this typically means that the duration of the activity is the more decreasing the more resources are being used.

The one-to-many resource role association of "load truck" with "wheel loaders" leads to a property *wheelLoaders* in the activity class *LoadTruck*, as shown in Figure 4. Notice that this property has the multiplicity 1..2, which means that it is collection-valued and has either a singleton or a two-element collection value.

When activities admit using more resources than required, this means they can be started as soon as the required number of resources are available, but they can also be started with a greater number of resources, typically implying a faster performance.

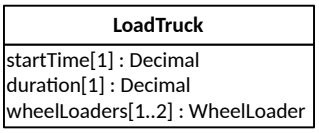

Figure 4: The class *LoadTruck* implementing the corresponding activity type.

2.1.2 Many-to-One Relationships

An example of a many-to-one relationship between activities and resources is the association between "examination" activities and "examination room" resources shown in Figure 5.

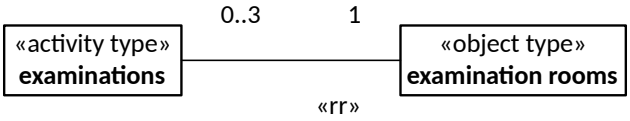

Figure 5: The loading of a truck requires at least one, and can be handled by at most two, wheel loaders.

In this example, examination rooms (as resources) admit of up to three examinations being performed at the same time. Such a *multitasking constraint* can be implemented with the help of a (multitasking) *capacity* attribute of the class implementing the resource object type. If such a constraint applies to all instances of a resource object type, *capacity* would be a class-level ("static") attribute, which is rendered underlined in a UML class diagram, as shown in Figure 6.

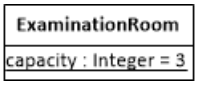

Figure 6: The class *ExaminationRoom* implementing the corresponding resource object type.

Alternatively, if different examination rooms have different capacities, then *capacity* would be an ordinary (instance-level) attribute of the class *ExaminationRoom*.

2.1.3 Many-to-Many Relationships

Examples of many-to-many relationships between activities and resources are rare and typically involve activities going on with interruptions. An example is the association between "teaching a course" activities and "teacher" resources shown in Figure 7. In this example, the teaching of a course admits at most two teachers who may be involved in at most 7 course teaching activities at the same time.

2.2 Principles of Resource Allocation

Both in reality and in simulation, the most important principle for allocating a human resource as the performer of a task is *role-based assignment* where a performer is selected from a pool of human resources that play a particular role in an organization, such as the role of an *order taker* in a sales department. Typically, human resources hold certain positions in an organization, and each *organizational position* implies one or more roles that a holder of that position may play. Consequently, *position-based* assignment is a form of role-based assignment.

By the nature of organizational roles, role-based assignment of human resources implies that the assigned resource has both the required capabilities/expertise and the permission/authorization for performing the given task. This seems to be overlooked by BPM research works that treat roles, capabilities and authorizations as separate issues (Arias et al. 2018; Pufahl et al. 2021).

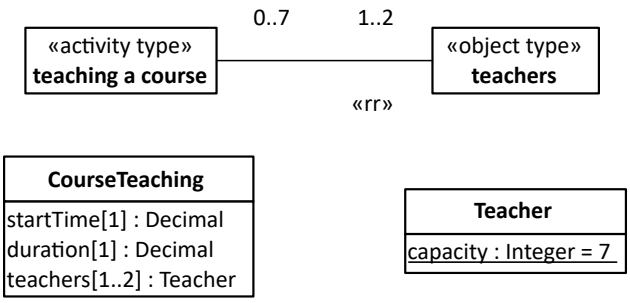

Figure 7: The teaching of a course is performed by teachers.

In simulation, we may assume that, by default, there is one resource pool for each position in an organization. However, in certain cases there may be two or more pools for an organizational position, possibly assigned to different organizational units.

In addition to this basic principle of role-based assignment, there are several important resource allocation approaches and policies:

Performer Task List
: Instead of allocating an available performer resource for immediately starting a task, it's also possible to assign a task to the task list of a performer resource and leave it to this performer when to start the task. For a BP simulation, this approach is more complex since it requires to model the decision making of the performer.

Persistent Resource Allocation
: When a follow-up activity requires a resource of the same type as its predecessor activity, it's a natural default to keep (or re-allocate) the same resource. A special case of this policy, limited to the human resource of a workflow task, has been called the *Retain Familiar* workflow pattern in (Russell et al. 2004).

Alternate Resource Pools
: Certain activities allow alternative resources, when no standard resources are available. For instance, in a pizza service company, when no order taker is available, and a new order call comes in, an available pizza baker can take the order. For instance, Tag (2015) explains that in the DES tool *ExtendSim*, a resource allocation policy specified as "1 RESOURCE FROM POOL A OR POOL B OR POOL C" means that there is a preferred pool A, and two alternate pools (B and C). A list of alternate resource pools may be associated with decreasing proficiency levels that increase the duration of activities and their rework probabilities.

Task Priorities
: Tasks may have different priorities, either defined by their type or individually. When several tasks compete for a resource, the resource is allocated to the task with the highest priority.

Activity Preemption
: An ongoing activity may be preempted by a higher-priority task requiring a resource used by the ongoing activity. In such a case, the activity is suspended and the required resource is released from it and allocated to the higher-priority task. The suspended activity is later resumed after a resource of the required type becomes available again.

Single-Step versus Multi-Step Allocation
: For each activity type, it has to be decided, if the resources required for a planned activity of that type are allocated at once or successively as soon as they become available. These two allocation policies can be called *Single-Step Allocation* and *Multi-Step Allocation*.

While in BPM research works such as (Russell et al. 2004; Pufahl et al. 2021), only *Performer Task List*, *Persistent Resource Allocation*, and *Task Priorities* have been considered, DES tools typically do not support *Performer Task List* and *Persistent Resource Allocation*, but all other policies.

2.2.1 Declarative versus Procedural Resource Allocation

DES tools still use the Seize-Delay-Release pattern introduced by GPSS (Gordon 1961), either explicitly or implicitly, for modeling resource-constrained activities. This modeling pattern represents a procedural modeling approach where all resource allocation and de-allocation steps have to be specified explicitly.

In the declarative resource modeling approach, activities have resource roles (special attributes for referencing the used resources), which are defined together with resource cardinality constraints. In this approach, (de-)allocation steps don't have to be specified explicitly in the simulation model. Rather, they are implied by the resource roles and their resource cardinality constraints such that the simulator can take care of performing the (de-)allocation steps.

In (Goel and Lin 2021), a procedural resource modeling approach for BPMN with explicit Seize-Release steps is proposed, abandoning BPMN's resource role concept. However, declarative modeling is superior in terms of model complexity, readability and maintainability.

2.3 Resource Failure/Repair Modeling

For various reasons, resources can fail: machines may break down, nurses or doctors may get sick. In addition, non-human resources may become unavailable due to scheduled maintenance, and human resources are not available when out-of-duty or off-shift.

In an event-based simulation approach, resource failures can be modeled with the help of two types of events: *failure* and *recovery*, and two related random variables: *failure recurrence* and *failure time*. In this approach, the next resource failure occurs after x time units where x is obtained from invoking the random variable sampling function `failureRecurrence()`. Each *failure* event triggers a *recovery* event, which is scheduled with a delay of y time units where y is obtained from invoking the random variable sampling function `failureTime()`.

In activity-based simulation, this simple approach can be refined by modeling the *repair* of a failed resource as an activity whose duration is provided by the random variable *repair time* (and which typically requires a repair person as a resource), such that the total failure time is the sum of the two random variables *repair lead time* (the time needed for getting a repair person to start the repair) and *repair time*.

A more advanced approach goes beyond the explicit modeling of repair activities, by introducing *implicit failed resource replacement* according to given redundancy policies, such as *hot* or *cold standby*, in which redundant resources that replace the failed primary resource are maintained active or are to be activated, respectively.

3 RESOURCE MODELING IN DISCRETE EVENT SIMULATION

In Discrete Event Simulation (DES), more precisely in the leading DES paradigm, called *Processing Network (PN)* simulation in Wagner (2022), a business process is modeled as a directed graph of event and activity nodes. Since all predominant DES tools, such as Arena, ExtendSim, Simio and AnyLogic, use the Seize-Delay-Release modeling pattern introduced by GPSS (Gordon 1961), each activity node is (explicitly or implicitly) preceded and succeeded by resource allocation (Seize) and de-allocation (Release) steps. In the definition of a Seize step, it can be specified that one or more resource objects from one or more resource pools are required. In addition, it is possible to specify alternate resource pools.

The DES/PN Seize-Release paradigm represents a procedural resource modeling approach, while the approach of BPMN based on resource roles represents a declarative resource modeling approach.

4 RESOURCE MODELING IN BUSINESS PROCESS MODELING

BPMN is the most widely used BP modeling language today (Weske 2012). Its popularity is due to its ease-of-use and versatility, supporting users with different backgrounds, from business analysts down to IT developers in charge of implementing BPs specified in BPMN.

Since BPMN has not been conceived with simulation in mind, a BPMN process model can only be turned into a simulation model by enriching it with additional model items, including case recurrence, activity durations and resource pools. For adding these and other items to a BPMN process model, the Workflow Management Coalition has proposed the *BPSim* standard (Workflow Management Coalition 2016).

4.1 Resource Modeling in BPMN

In BPMN, an activity may have a set of *resource roles*, provided by the *resources* attribute. The BPMN specification allows "a specific individual, a group, an organization role or position, or an organization" to be assigned via this attribute.

In general, however, a resource role is specified by means of a role name (like *order taker*) and an organizational position (like *sales clerk*), such that resources can be allocated to an activity for this role if they hold this position.

BPMN process diagrams use swimlanes for associating an activity type with a performer resource role, but this visual syntax does not allow associating more than one resource role. In particular, additional passive resource types (like rooms or machines/devices/tools) cannot be associated with an activity using the visual syntax of swimlanes. Thus, while more than one resource roles of an activity can be specified in a BPMN model (with the help of the attribute *resources*), only the performer role can be visually specified in the corresponding process diagram.

BPMN does neither support the specification of resource cardinality constraints nor of multi-tasking constraints.

As pointed out in (Goel and Lin 2021), BPMN has not found much recognition in the operations management and operations research communities due to its limited support of resource modeling requirements.

4.2 Resource Modeling in BPSim

The *BPSim* standard (Workflow Management Coalition 2016) allows specifying resource roles (with the `selection` and `role` parameters), the size of resource pools (with the `quantity` parameter), and the `availability` of resources (e.g., based on calendars). When resource roles are specified they override any resource role definition in the underlying BPMN model.

BPSim does, however, neither support resource cardinality constraints, nor multitasking constraints.

5 RESOURCE MODELING IN OEM&S AND DPMN

The *Discrete Event Process Modeling Notation (DPMN)* is based on *Object Event (OE) Modeling and Simulation (M&S)*, which is a new DES paradigm based on the three most important ontological categories: *objects*, *events* and *causal regularities*. The formal semantics of OEM&S defined in (Wagner 2017) provides a general semantics for classical *Event Graphs* (Schruben 1983) and for *Object Event Graphs*.

Since OEM&S accommodates the concepts of resource-constrained activities and processing activities, it integrates important DES concepts and supports modeling general forms of (BPMN-style) *Activity Networks* and (GPSS-style) *Processing Networks*, which extend Activity Networks by adding the concept of *processing objects* flowing through the network. The semantics of DPMN Activity Networks is obtained by reduction to Object Event Graphs using two rewrite patterns, and the semantics of DPMN Processing Networks is obtained by reduction to Activity Networks.

It has been an unfortunate course of the history of science that Event Graphs have not become more well-known and, instead of Event Graphs, Petri Nets have been chosen as the standard semantics of BP models, despite the fact that Event Graphs would have been a much more natural choice.

OEM&S combines Object-Oriented (OO) Modeling with the event scheduling paradigm of Event Graphs (Schruben 1983). The relevant types of objects, events and activities are described in an information model, which is the basis for making a process model. The preferred form of information models are *OE Class Models* (a form of UML Class Models), while the preferred form of process models are DPMN Process Diagrams.

In OEM&S, object, event and activity types are modeled as special categories of classes in an OE Class Diagram. Random variables are modeled as a special category of class-level operations constrained to comply with a specific probability distribution such that they can be implemented as static methods of a class. Finally, *causal regularities* are modeled in the form of *event rules* in DPMN process diagrams, such that they can be implemented as special `onEvent` methods of event classes.

Resources and the dependency of activities on them are modeled in an OE class model, which serves as the basis of a DPMN process model. Resources are modeled as object types in the form of class rectangles, while the dependency of an activity on a resource is modeled as an association line connecting the activity type rectangle with the resource type rectangle, as shown in Figure 8.

For making process models more self-contained, it's also an option to model resources and the dependency of activities on them in a DPMN process model, as shown in Figure 9. In such a process model, (human) performer resources are represented as compartments (called 'swimlanes') containing the activities being performed by them, while additional passive resources are associated with activities by means of a connection line with a square dot at the end of the resource rectangle.

5.1 Case Study

We consider a pizza service company that takes phone orders for making and delivering pizzas, with the help of order takers, pizza makers, ovens and a crew of pizza delivery scooter drivers.

For getting a quick impression, you can run this model implemented with the JavaScript-based OEM&S framework OESjs from the Sim4edu website.

5.1.1 Modeling Object, Event and Activity Types

In OEM&S, object, event and activity types are modeled as special categories of classes in an OE class diagram like the one shown in Figure 8, which defines:

1. the resource roles *OrderTaker*, *PizzaMaker*, *Oven* and *ScooterDriver* object types *OrderTaker*, *PizzaMaker*, *Oven* and *ScooterDriver*;
2. the event type *OrderCall*;
3. the activity types *TakeOrder*, *MakePizza* and *DeliverPizza*.

In addition to the object, event and activity type rectangles, the OE class diagram also includes resource role association lines between activity type rectangles and resource object type rectangles with multiplicities at both ends representing resource cardinality constraints and multitasking constraints.

For instance, the association line between *MakePizza* and *PizzaMaker* represents a resource dependency with a resource role *pizzaMakers* having the resource cardinality constraint of "exactly 2" pizza makers that need to be allocated to a *MakePizza* activity, and a multitasking constraint of "at most 1" *MakePizza* activities, in which a pizza maker can participate at the same time.

The association line between *MakePizza* and *Oven* represents an additional resource dependency with a resource role *oven* having the resource cardinality constraint of "exactly 1" oven that needs to be allocated to a *MakePizza* activity, and a multitasking constraint of "at most 2" *MakePizza* activities, in which an oven can participate at the same time.

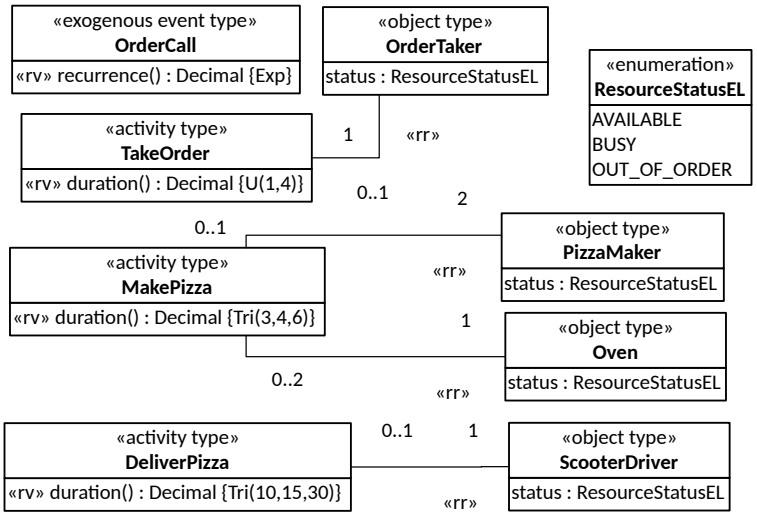

Figure 8: The OE class model.

5.1.2 Modeling the Flow of Events and Activities

In OEM&S, the flow of events and activities, that is, the dynamics of a system, is modeled in a DPMN process diagram like the one shown in Figure 9, which defines the possible sequences of events and activities with the help of *resource-dependent activity scheduling* arrows having three bars for symbolically representing the involved task queues.

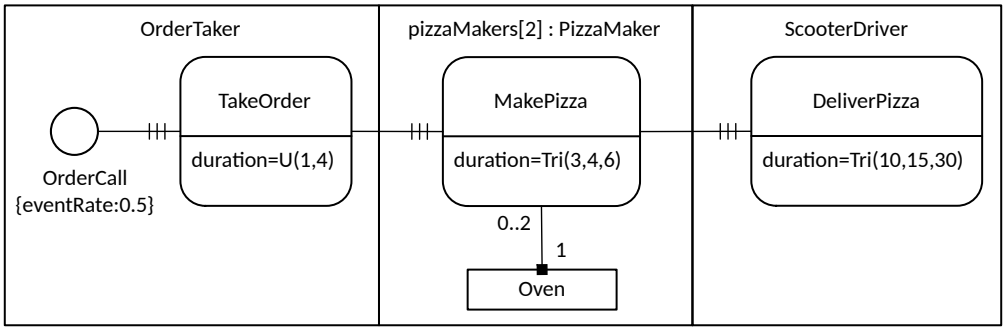

Figure 9: The DPMN process model.

This model defines the following three event rules:

1. WHEN an *OrderCall* event occurs, if an order taker is available, THEN allocate him and start a new *TakeOrder* activity, OTHERWISE enque a new *TakeOrder* task, for which an order taker still has to be allocated.
2. WHEN a *TakeOrder* activity end event occurs, if two pizza makers and an oven space are available, THEN allocate them and start a new *MakePizza* activity, OTHERWISE allocate the available resources and enque a new *MakePizza* task, for which the remaining resources still have to be allocated.
3. WHEN a *MakePizza* activity end event occurs, if a delivery scooter driver is available, THEN allocate her and start a new *DeliverPizza* activity, OTHERWISE enque a new *DeliverPizza* task, for which a delivery scooter driver still has to be allocated.

Notice that, for being self-contained, the process model repeats the specification of resource role associations that have already been specified in the underlying OE class model shown in Figure 8.

Notice also that the process model does not include any element representing a resource pool. It is assumed that for any organizational position and for any passive resource type described in the underlying OE class model (here: *OrderTaker*, *PizzaMaker*, *ScooterDriver* and *Oven*), the organization under consideration has a corresponding resource pool. By default, each resource role of an activity type is associated with a resource pool having the same (yet pluralized) name, such that its resource objects are instances of a corresponding resource role type, which is an organizational position in the case of human resources.

For instance, for the *MakePizza* activity a pool *ovens* is assigned to its resource role *oven* by default. The members of the pool *ovens* are instances of the (resource) object type *Oven*. Likewise, a pool *pizzaMakers* is assigned to the *MakePizza* resource role *pizzaMaker*. The members of this pool are instances of the organizational position *PizzaMaker*. These default pool assignments are normally not shown in a DPMN diagram, but an OE simulator takes care of them.

Combined with its underlying OE class design model, a DPMN process design model provides a computationally complete specification of a simulation model.

5.1.3 Modeling an Alternate Resource Pool

At times, the order takers cannot cope with the number of incoming calls, and the waiting line gets too long such that customers grow impatient. In order to decrease this risk, the pizza service company adds the resource role "order taker" as a second role to the organizational position *PizzaMaker*, such that not only the *OrderTaker*, but also *PizzaMaker* employees can be allocated to *TakeOrder* tasks. This can be expressed in an OE class model with help of a special generalization arrow defining *PizzaMaker* to be a subclass of *OrderTaker* as shown in Figure 10.

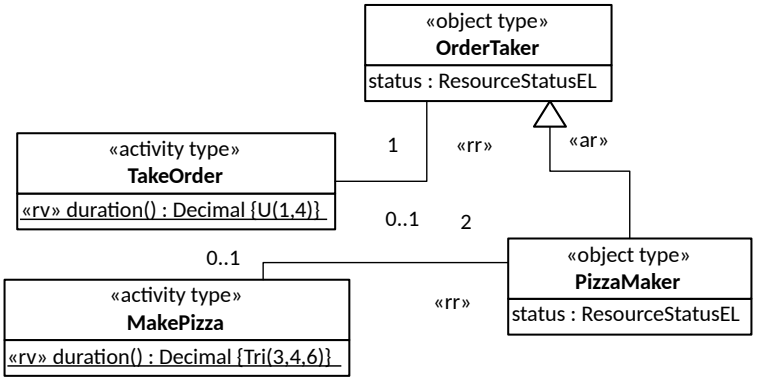

Figure 10: Making *PizzaMaker* an alternate resource pool for *TakeOrder* tasks.

The alternate resource pool definition of Figure 10 means that when a new order call comes in and no member of the *OrderTaker* pool is available, an available member of the *PizzaMaker* pool will be allocated to handle the call. This order-maximizing policy can be further strengthened by:

1. defining a higher *task priority* for *TakeOrder* tasks, which implies that members of the *PizzaMaker* pool when they are de-allocated from a *MakePizza* activity, are not allocated to the next *MakePizza* task, but rather to the next *TakeOrder* task;
2. setting *activity preemption* for *MakePizza* activities, which means that when a new order call comes in and all members of the the *OrderTaker* and *PizzaMaker* pools are busy, then one of the *MakePizza* activities is suspended for allocating its *PizzaMaker* resource to the call.

5.1.4 Summary of Case Modeling Requirements

The requirements of the *Pizza Service* case study are summarized as follows:

- Incoming calls arrival rate shall be defined as a random variable with a given probability distribution function (e.g., exponential).
- *Take Order* duration shall be defined as a random variable with a given probability distribution function (e.g., uniform).
- *Make Pizza* and *Deliver Pizza* duration shall be defined as a random variable with a given probability distribution function (e.g., triangular).
- The Pizza Service process shall consider pools of resources: 2 order takers, 10 pizza makers, 5 ovens and 20 scooter drivers.
- Resources failure ad repair occurrences shall be defined as random variables with a given probability distribution function (e.g., exponential).

6 RESOURCE MODELING IN PyBPMN/eBPMN

This Section introduces an approach, denoted as *PyBPMN/eBPMN*, for *modeling* and *simulation-based analysis* of BPs. The approach makes use of *PyBPMN (Performability-enabled BPMN)*, a BPMN extension that addresses the specification of performance and reliability properties of BPs (Bocciarelli and D'Ambrogio 2011b; Bocciarelli and D'Ambrogio 2011a; Bocciarelli and D'Ambrogio 2014; Bocciarelli et al. 2014; D'Ambrogio et al. 2016) and *eBPMN*, a Java-based domain-specific simulation framework introduced in (Bocciarelli et al. 2014).

6.1 Performability-enabled BPMN

The PyBPMN extension has been based on principles and standards introduced in the model-driven engineering field, specifically by the OMG's Model Driven Architecture (MDA) (Ojbect Management Group 2003). Such an extension addresses the non-functional characterization of a BP according to four different dimensions:

- **workload**, to model the workload for the tasks in the process;
- **performance**, to specify the performance properties, i.e., efficiency-related properties such as the service time, associated to single tasks;
- **reliability**, to specify the reliability properties of the resources that tasks depend on to carry-out the work requests;
- **resource management**, to describe the execution platform actually used for executing the BP.

Figure 11 shows the *PyBPMN metamodel*, i.e., a class diagram in which classes define constructs to be instantiated at model level. The figure also shows the relations between novel *PyBPMN metaclasses* (i.e., classes in the metamodel) and the extended BPMN ones.

The PyBPMN-based modeling approach doesn't introduce a separate notation for representing the BP under study. Rather, the functional requirements of the process to be analyzed are initially used for specifying a standard BPMN model. Then, the model is enriched with annotations that capture the non-functional requirements, according to elements (or metaclasses) introduced in the PyBPMN metamodel.

As an example, for describing the performance characterization of a BP, PyBPMN provides the abstract metaclass `PaQualification`, that can be specialized as `PaService` for specifying the *service demand* for executing a single request. As regards the representation of BP resources reliability, the PyBPMN metamodel includes the abstract metaclass `DaQualification`, which can be specialized as `DaFailure`, for specifying the failure-related properties, e.g., the probability distribution of a failure or a repair event.

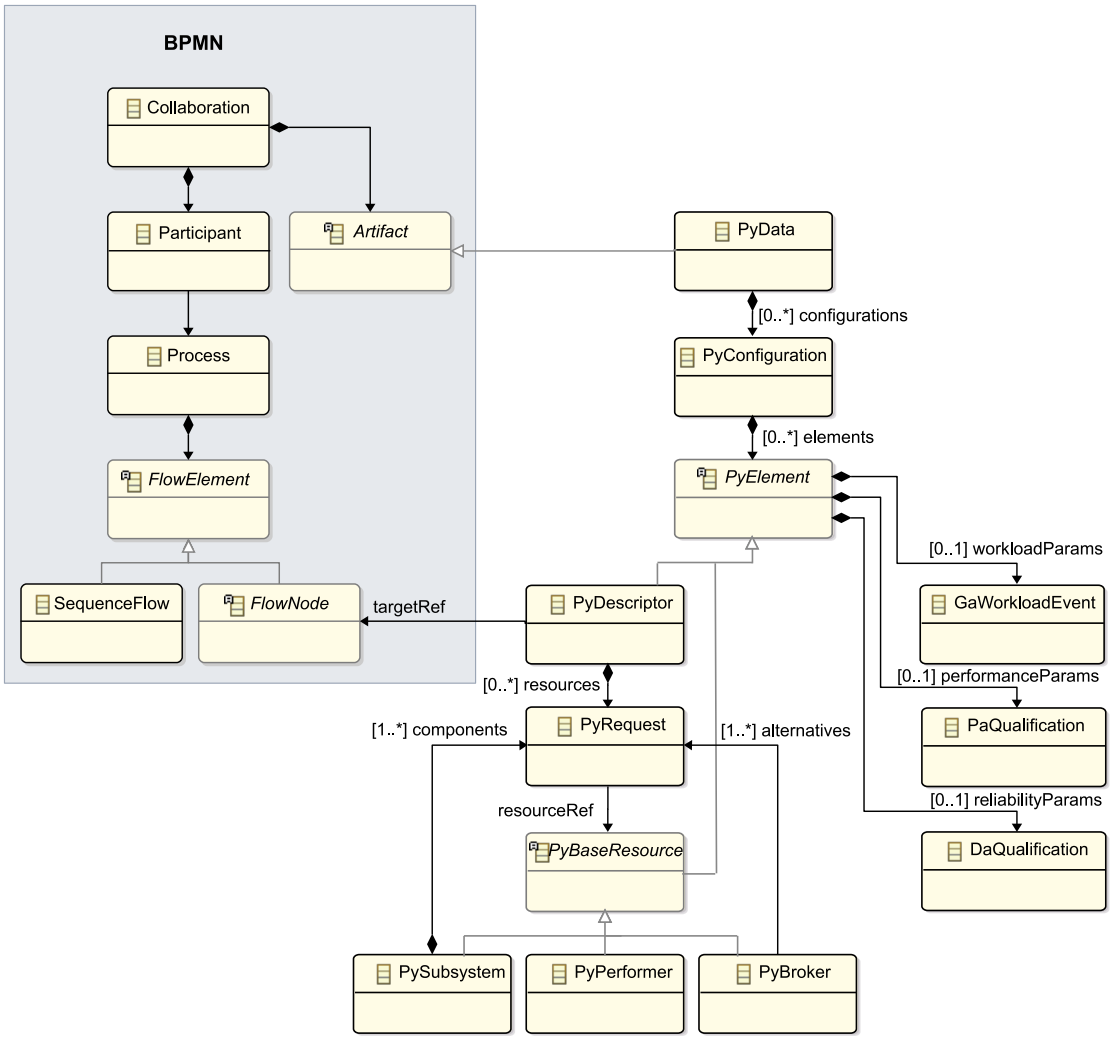

Figure 11: Relations between BPMN and PyBPMN metaclasses.

A detailed description of the PyBPMN extension is given in (Bocciarelli and D'Ambrogio 2011b; Bocciarelli and D'Ambrogio 2011a; Bocciarelli and D'Ambrogio 2014; D'Ambrogio et al. 2016).

PyBPMN provides a resource definition which is complementary to the standard BPMN one. In BPMN, a resource is an abstract role involved in an activity, whereas the PyBPMN resource definition allows the specification, at design time, of 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.

As an example, the abstract class `PyBaseResource` represents a resource that is part of the execution platform. It can be specialized in terms of the concrete classes `PyPerformer`, `Broker` and `PySubsystem`. The `PyPerformer` class represents the actual work performer that executes the service requests by selecting the required number of performers from a pool of available resources. The `Broker` class manages the selection of a resource from a set of candidate resources according to a given *policy* (e.g., *hot* or *cold standby*). Finally, the `PySubsystem` class allows the modeling of a complex resource defined in terms of its elementary components.

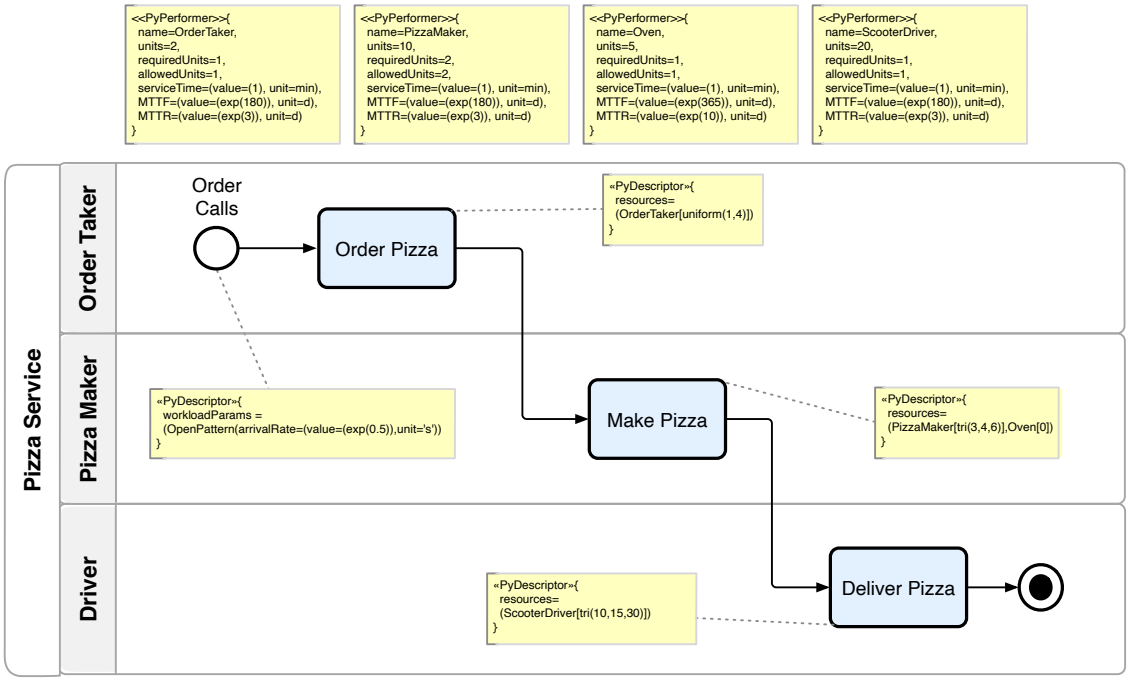

Figure 12: Pizza Service process specified as an annotated BPMN Model.

6.2 eBPMN Simulation Framework

PyBPMN models can be directly simulated by using eBPMN, a Java-based domain-specific simulation framework introduced in (Bocciarelli et al. 2014). The eBPMN architecture, which has been designed to comply with the execution semantics of the BPMN version 2.0 specification, reflects the structure of the PyBPMN metamodel. Relevant concrete classes have been defined for explicitly handling the non-functional characterization of a BP (e.g., the service demand for a given resource or the probability of a failure/repair event), in order to mimic the realistic behavior of the BP under study. As an example, `EBPMNResource` is the abstract superclass for all of the resource classes defined in eBPMN. This class holds a list of references to the eBPMN elements that use the resource. The `EBPMNResource` class can be specialized as *Performer*, *Broker*, or *Subsystem*, which provide the implementation of the relevant PyBPMN metamodel elements. Specifically, the `Performer` is the entity that actually executes the service request.

In order to show how PyBPMN/eBPMN can be effectively used for modeling and then simulating a BP, an example application to the Pizza Service process introduced in 5.1 is presented in the next section.

6.3 Case Study

As stated in Section 6.1, the PyBPMN/eBPMN approach does not introduce a separate notation. Rather, it is based on the initial specification of a standard BPMN model of the BP under study, according to its functional requirements.

Then, the BPMN model has to be enriched with textual annotations, according to the non-functional requirements characterizing the process in terms of the workload, the service demand for each activity and the description of the underlying execution platform.

The annotated BPMN model is provided in Figure 12, in which specific PyBPMN annotations are introduced for addressing the requirements summarized in Section 5.1.4.

The arrival rate for order calls is specified by use of the `<<PyDescriptor>>` annotation including the attribute *arrivalRate=(value=(exp(0.5))*.

Table 1: Feature Comparison.

Feature	BPMN	DES/PN	DPMN	eBPMN/PyBPMN
Declarative res. roles	+	−	+	+
Res. min card constraints	−	+	+	+
Res. max card constraints	−	−	+	+
Multitasking constraints	−	+	+	+
Alternate resource pools	−	+	+	+
Task priorities	−	+	−	−
Activity preemption	−	+	−	−
Single/multi-step allocation	−	+	−	−
Persistent allocation	−	n/a	+	+
Resource failure modeling	−	+	+	+
Implicit resource replacement	−	−	−	+

`<<PyPerformer>>` annotations specify the characteristics of resources composing the execution platform: the human resources `OrderTaker`, `PizzaMaker` and `Driver`, and the resource `Oven`. Each annotation describes the *service time* (i.e., the time required for the execution of a single request), the resource pool configuration (attributes *units*, *requiredUnits* and *allowedUnits*) and the reliability characterization in terms of the probability distribution function associated to failure and repair events (*MTTF*and *MTTR* attributes, respectively).

`<<PyDescriptor>>` annotations attached to BPMN *activity* elements specify the service demand for each required resource. As an example, for executing the *Take Order* activity the *OrderTaker* human resource is required and the service demand is defined as a *Uniform* random variable. The *Make Pizza* activity requires two resources, the *PizzaMaker* and the *Oven*. It should be noted that the *PizzaMaker* is busy during the time required for both *preparing* and *cooking* a pizza, and that both the *PizzaMaker* and the *Oven* resources must be in a *working* state. Thus, even though the *Make Pizza* activity requires two resources, the *Oven* service demand is assumed to be 0, being the whole service demand allocated to the *PizzaMaker*.

Once the annotated BPMN model has been specified, a chain of *model transformations* is introduced to enact the BP actual simulation. Specifically, two transformations are executed. The first one is the `BPMN-to-PyBPMN` *model-to-model* transformation, which takes as input the annotated BPMN model (serialized to an XML file) and yields as output an XMI-based PyBPMN model. The second one is the `PyBPMN-to-eBPMN` *model-to-text* transformation, for generating the corresponding eBPMN implementation.

A complete description of the above-mentioned model transformations can be found in (Bocciarelli et al. 2019).

7 COMPARISON

Table 1 summarizes the main features offered by the BP modeling and simulation approaches described in Section 5 and 6, as compared to conventional approaches based on the BPMN standard and the DES processing network (PN) paradigm. A + symbol denotes a provided feature, a − symbol denotes a missing feature, and 'n/a' denotes 'not applicable'.

8 CONCLUSIONS

Modeling the management of resources in BP modeling and simulation is a complex issue. The establishment of a conceptual framework for resource modeling has been hampered by the schism between the two scientific areas involved, DES and BPM.

In this paper, we have summarized the most important concepts for resource modeling and discussed how resources are modeled in BPMN and in DES in general, and in two particular approaches: OEM&S/DPMN and PyBPMN/eBPMN.

REFERENCES

Arias, M., J. Munoz-Gama, and M. Sepúlveda. 2018. "Towards a Taxonomy of Human Resource Allocation Criteria". In *Business Process Management Workshops*, edited by E. Teniente and M. Weidlich, 475–483. Cham: Springer International Publishing.

Bocciarelli, P., and A. D'Ambrogio. 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, 160–168. Boston, Massachusetts.

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, 1–36. New York: 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*:573–598.

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*, MODELSWARD'14. Lisbon, Portugal: SciTePress.

Bocciarelli, P., A. D'Ambrogio, A. Giglio, and E. Paglia. 2019. "BPMN-Based Business Process Modeling and Simulation". In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K.-H. G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, 1439–1453. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

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*. Pasadena, CA, USA.

Goel, A., and M.-B. Lin. 2021. "Resource Requirements in Business Process Modelling from an Operations Management Perspective". SSRN: https://ssrn.com/abstract=3869592.

Gordon, G. 1961. "A General Purpose Systems Simulation Program". In *Proceedings of the Eastern Joint Computer Conference: Computers - key to Total Systems Control*, edited by W. H. Ware, 87–104. New York: Association for Computing Machinery.

Ojbect Management Group 2003. "MDA Guide, Revision 2.0 (ormsc/14-06-01)". https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf, accessed 2022-04-15.

Ojbect Management Group 2014. "Business Process Model And Notation (BPMN) Version 2.0.2". https://www.omg.org/spec/BPMN/, accessed 2022-04-15.

Pufahl, L., S. Ihde, F. Stiehle, M. Weske, and I. Weber. 2021. "Automatic Resource Allocation in Business Processes: A Systematic Literature Survey". Preprint, https://doi.org/10.48550/arXiv.2107.07264.

Russell, N., A. H. ter Hofstede, D. Edmond, and W. M. van der Aalst. 2004. *Workflow Resource Patterns*. http://www.workflowpatterns.com/patterns/resource/, accessed 2022-04-15.

Schruben, L. W. 1983. "Simulation Modeling with Event Graphs". *Communications of the ACM* 26:957–963. https://dl.acm.org/citation.cfm?id=358460.

Tag, P. H. 2015. "Improving Business Project Performance by Increasing the Effectiveness of Resource Capacity and Allocation Policies". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 856–867.

Wagner, G. 2017. "An Abstract State Machine Semantics for Discrete Event Simulation". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page: Piscataway, New Jersey. Institute of Electrical and Electronics Engineers (IEEE), Inc.

Wagner, G. 2022. *Business Process Modeling and Simulation with DPMN*. E-Book, https://sim4edu.com/reading/bpms-dpmn/, accessed 2022-04-15.

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

Workflow Management Coalition 2016. "Business Process Simulation Specification Version 2.0". https://www.bpsim.org/specifications/2.0/WFMC-BPSWG-2016-01.pdf, accessed 2022-04-15.

AUTHOR BIOGRAPHIES

PAOLO BOCCIARELLI is a postdoc researcher in the Department of Enterprise Engineering, University of Rome Tor Vergata, Italy. His email address is paolo.bocciarelli@uniroma2.it.

ANDREA D'AMBROGIO is Associate Professor of Systems and Software Engineering in the Department of Enterprise Engineering, University of Rome Tor Vergata, Italy. His email address is dambro@uniroma2.it.

GERD WAGNER is Professor of Internet Technology in the Department of Informatics, Brandenburg University of Technology, Germany. His email address is G.Wagner@b-tu.de.