Proceedings of the Operational Research Society Simulation Workshop 2023 (SW23)
C. Currie and L. Rhodes-Leader, eds.

Preprint

BUSINESS PROCESS MODELLING AND SIMULATION WITH DPMN, ANYLOGIC AND SIMIO – A TUTORIAL

Dr. Gerd Wagner

Brandenburg University of Technology
Konrad-Wachsmann-Allee 5
03046 Cottbus, GERMANY
g.wagner@b-tu.de

ABSTRACT

Business process modelling and simulation has been a core research topic both in the area of Discrete Event Simulation and in the area of Business Process Management for a long time. However, both areas have largely ignored each other's research results. In particular, Discrete Event Simulation research has not managed to establish a general conceptual foundation and modelling language for Processing Networks, so vendors are using their own proprietary terminology and diagram language in their "process modelling" tools, ignoring the process modelling language BPMN, which has been established as a widely adopted standard in Business Process Management.

In this tutorial, we present the conceptual foundations of, and a modelling language for, Activity Networks and Processing Networks. The Discrete Event Process Modelling Notation (DPMN) proposed by Wagner (2018) combines the visual syntax of BPMN with the rigorous semantics of Event Graphs (Schruben 1983). DPMN allows making platform-independent visual simulation models that can be implemented with Discrete Event Simulation platforms such as AnyLogic or Simio.

Keywords: Business Process Modelling, BPMN, DPMN, AnyLogic, Simio

1 INTRODUCTION

Business processes have been modelled 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) modelling and simulation.

While Schruben (1983) has proposed *Event Graphs* as a visual modelling language for the DES paradigm of *Event-Based Simulation (ES)*, no common modelling language has been established for the DES paradigm of *Processing Networks* (often called "process modelling"). Each "process modelling" tool (such as Arena, Simio, AnyLogic, etc.) is using their own proprietary terminology and diagram language. In BPM, the BP modelling notation BPMN, officially called 'Business Process Model and Notation' (OMG 2014), has been established as a widely adopted standard. BPMN has been largely ignored in DES, while Event Graphs have been ignored in BPM.

The term *Discrete Event Simulation* has been established as an umbrella term subsuming various kinds of computer simulation approaches, all based on the general idea of modelling the dynamics of a discrete system as a series of (explicit or implicit) events that change the system's state over time.

In the DES literature, it is often stated that DES is based on "entities flowing through a system". While this narrative applies to the DES paradigm of Processing Networks, called "process modelling" by Pegden (2010), it characterizes a special (yet important) kind of DES only, and it does not apply to all discrete event systems. The "process modelling" paradigm should be better called *Processing Network (PN)* paradigm, since it is not about process modelling in general, but only about modelling a particular kind of discrete processes that happen in Processing Networks (which generalize *Queuing*

Networks). It has been pioneered by GPSS (Gordon 1961) and SIMAN/Arena (Pegden and Davis 1992) and is implemented in various forms by all modern off-the-shelf simulation tools, including Simio and AnyLogic.

1.1 Some Remarks on the History of DES

Pegden (2010) explains that the history of DES has been shaped by three fundamental paradigms: Markowitz, Hausner, and Karr (1962) pioneered *Event-Based Simulation (ES)* with *SIMSCRIPT*, Gordon (1961) pioneered *Processing Network Simulation (PNS)* with *GPSS*, and Dahl and Nygaard (1967) pioneered *Object-Orientation (OO)* and the computational concept of co-routines for asynchronous programming with their simulation language *Simula*.

Notice, however, that OO does not represent a DES paradigm, but rather an information/data modelling paradigm for conceptual modelling and software design modelling, as well as a programming paradigm. In fact, the Simula paradigm is characterized by a combination of OO modelling and using co-routines for implementing the dynamics of a discrete system without an explicit computational concept of events in a way that is sometimes called 'process interaction' approach.

According to Pegden (2010), ES has been widely used during the first 20 years of simulation, due to its great flexibility allowing to efficiently model a wide range of complex systems. Later, however, the PNS paradigm, implemented by tools like Arena, Simul8, FlexSim, Simio and AnyLogic, became the dominant approach in practical applications of simulation because it is based on the higher-level concept of processing activities and allows no-code (or low-code) simulation engineering with graphical user interfaces and appealing visualizations.

After OO had been established as the predominant paradigm in software engineering in the 1990s, it was also adopted by many simulation tools, which weaved it into their PNS approach. Pegden (2010) remarks that "Many process and object based simulation tools maintain an event capability as a 'backdoor' for flexibility". Consequently, modern DES tools allow combining PN models with OO modelling and event scheduling.

As argued by Pegden (2010), ES is the most fundamental DES paradigm since the other paradigms also use events, at least implicitly. However, Pegden does not explain in which way the other paradigms are built upon ES.

1.2 Fragmentation and Conceptual Confusion in DES

Today, after a history of more than 50 years, the field of DES is fragmented into many different paradigms and formalisms, based on different concepts and terminologies. Unlike other scientific fields, it didn't achieve much conceptual unity regarding its foundations, which would be crucial for facilitating scientific progress.

There is a lot of conceptual confusion in the field of DES. For instance, Banks et al (2010) define that DES is "the modelling of systems in which the state variable changes only at a discrete set of points in time". Remarkably, this definition does not even mention the concept of events, possibly for accommodating approaches that do not explicitly refer to events, such as Petri Nets, Activity Cycle Diagrams and DEVS.

While some authors, such as Pegden (2010), distinguish between three fundamental DES approaches: ES, PNS and OO, others, such as Banks (1998), distinguish between four approaches: ES, PNS (called "process-interaction"), Activity Scanning and the Three-Phase Method. This disagreement about fundamental concepts, and the different terms used by different authors for the same concepts (e.g., the PNS paradigm has the following names in the DES literature: "process-oriented", "process-based", "process-interaction", "process-centric"), must be very confusing for students of and beginners in DES.

We follow the view of Pegden (2010) that ES is the most fundamental DES paradigm, although in many DES textbooks, e.g., in (Banks et al 2010), this is not explained. Adopting ES as the most fundamental DES paradigm implies that other paradigms should extend it in a conservative manner such that its basic concepts (and their semantics) are preserved.

The lack of a scientifically established conceptual foundation of DES and an accompanying standard terminology is witnessed in the diversity of terminologies and diagram languages used in DES

software packages. Typically, DES practitioners are locked into the terminology (and implementation idiosyncrasies) of the simulation platform they use, and often not aware of the general, platform-independent and implementation-agnostic concepts.

The use of proprietary terminologies and diagram languages makes it hard for simulation beginners to learn how to use a tool and for expert users of a tool to switch, or interchange models, from their tool to another one. Notice especially the strange term "Agent" used by AnyLogic, instead of the Arena term "Entity", for processing objects like manufacturing parts in production systems or patients in hospitals. It is confusing to call a manufacturing part, such as a wheel in the production of a car, an "agent".

1.3 Object Event Modelling and Simulation

In (Wagner 2018; 2020; 2021), we show how to extend ES by adding concepts like objects and activities resulting in *Object Event Modelling and Simulation (OEM&S)*, a new general DES paradigm based on the two most important ontological categories: objects and events (Guizzardi and Wagner 2010).

OEM&S combines OO modelling with the event scheduling approach of ES. *Object Event Simulation (OES)* is a conservative extension of ES. While ES defines the system state structure in the form of a set of global variables, OES defines it in the form of a set of objects (or object states), such that their attributes take the role of state variables.

In (Wagner 2018), we have introduced a variant of BPMN, called *Discrete Event Process Modelling Notation (DPMN)*, and have shown how an OEM approach based on UML Class Diagrams and DPMN Process Diagrams allows defining a set of object types OT, a set of event types ET, and a set of event rules R. In (Wagner 2017), we have shown that (a) these three sets define a state transition system, where the state space is defined by OT and ET, and the transitions are defined by R, and (b) such a transition system represents an *Abstract State Machine* (Gurevich 1985). This fundamental characterization of an OES model provides a formal semantics for OES.

Since OEM&S accommodates the concepts of resource-constrained activities and processing activities, it integrates important DES concepts and supports modelling general forms of (BPMN-style) *Activity Networks* and (GPSS-style) *Processing Networks*, which extend Activity Networks by adding processing objects flowing through the network.

1.4 Discrete Processes and Business Processes

A **discrete event process**, or simply *discrete process (DP)*, consists of a partially ordered set of **events** such that each of them causes zero or more discrete state changes of affected **objects**. When two or more events within such a process have the same order rank, this means that they occur simultaneously. A discrete process may be an instance of a *discrete process type* defined by a **discrete process model**.

A **business process** (BP) is a discrete process that involves **activities** performed by *organizational agents* qua one of their *organizational roles* defined by their *organizational position*. Typically, a business process is an instance of a business process type defined by an organization or organizational unit (as the owner of the business process type) in the form of a **business process model**.

While there are DPs that do not have an organizational context (like, for instance, message exchange processes in digital communication networks or private conversations among human agents), a BP always happens in the context of an organization.

The performance of a resource-dependent activity is constrained by the availability of the required **resources**, which may include *human resources* or other resource objects (such as rooms or devices).

There are two kinds of business process models:

1. BPMN-style **Activity Networks** (ANs) consisting of *event nodes* and *activity nodes* (with task queues) connected by means of **event scheduling arrows** and **resource-dependent activity scheduling (RDAS) arrows**, such that event and activity nodes may be associated with objects representing their participants. In the case of an activity node, these participating objects include the resources required for performing an activity. Typically, an activity node is associated with a particular resource object representing the activity **performer**.
2. GPSS/Arena-style **Processing Networks** (PNs) consisting of *entry nodes*, *processing nodes* (with task queues and input buffers) and *exit nodes* connected by means of **processing flow**

arrows, which overlay an RDAS arrow with an **object flow arrow**. The PN concept is a conservative extension of the AN concept, that is, a PN is a special type of AN.

In an AN, all activity nodes have a task queue filled with tasks (or planned activities) waiting for the availability of the required resources. An RDAS arrow from an AN node to a successor activity node expresses the fact that a corresponding activity end event (or plain event) triggers the deferred scheduling of a successor activity start event, corresponding to the creation of a new task in the task queue of the successor activity node.

A **workflow model** is a BP model that only involves performer resources (typically human resources). Examples of industries with workflow processes are insurance, finance (including banks) and public administration. Most other industries, such as manufacturing and health care, have business processes that also involve non-performer resources or processing objects.

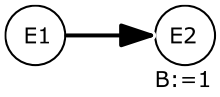
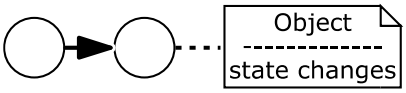
Business process (BP) models focus on describing the possible sequences of events and activities, based on conditional and parallel branching, but they also describe the dependencies of activities on resource objects, either *declaratively*, as in BPMN and DPMN, by defining resource roles for activities,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

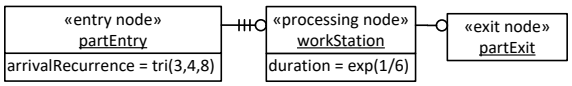
1.5 Processing Networks Generalize Queuing Networks

A **PN process** is a business process that involves one or more *processing objects* and includes *arrival events*, *processing activities* and *departure events*. An arrival event for one or more *processing objects* happens at an *entry station*, from where they are routed to a *processing station* where *processing activities* are performed on them, before they are routed to another processing station or to an *exit station* where they leave the system via a departure event.

A **PN process model** defines a PN where each node represents a combination of a spatial object and an event or activity variable:

1. Defining an **entry node** means defining both an *entry station* object (e.g., a reception area or a factory entrance) and a variable representing *arrival* events for arriving *processing objects* (such as people or manufacturing parts).
2. Defining a **processing node** means defining both a *processing station* object (often used as a resource object, such as a workstation or a room) and a variable representing *processing activities*.
3. Defining an **exit node** means defining both an *exit station* object and a variable representing *departure* events.

	Layer	Elements/Concepts	Diagrams
Event-Based Simul.	**Event Graphs** (Schruben 1983)	Event Circles, Event Scheduling Arrows, Variable Value Assignments	
Object Event Modelling and Simulation (OEM&S)	**Object Event Gra**phs (Basic DPMN)	+ Objects w/ State Changes	
	Activity Networks (DPMN-A)	+ Activities + Resource Roles + Resource Cardinality Constraints + Resource Pools + Resource-Dependent Activity Scheduling Arrows	

	Processing Networks (DPMN-PN)	+ Processing Activities + Entry/Processing/Exit Nodes + Processing Flow Arrows	

In a PN, all processing nodes have a task queue and an input buffer filled with processing objects that wait to be processed. A PN where all processing activities have exactly one abstract resource (often called a "server") is also known as a **Queuing Network** in *Operations Research* where processing nodes are called "servers" and processing objects are called "entities" or "jobs".

2 FROM EVENT GRAPHS VIA ACTIVITY TO PROCESSING NETWORKS

In this section, summarizing (Wagner 2018; 2020; 2021), we show how DPMN is constructed by incrementally extending Event Graphs by adding increasingly high-level modelling concepts: in the first step, we add the concept of objects, resulting in *Object Event Graphs*; in the second step, we add the concept of (resource-constrained) activities, resulting in *Activity Networks*; and in the third step we add the concept of processing activities, resulting in *Processing Networks*. Object Event Graphs, Activity Networks and Processing Networks are special forms of DPMN process models.

2.1 From Event Graphs to Object Event Graphs

Event Graphs define graphically, which state changes and follow-up events are triggered by an event. The Event Graph shown in Figure 1 models a manufacturing workstation, which is part of a manufacturing business system (viewed as a basic queuing system in *Operations Research*). It defines (a) two state variables: *L* for the length of an arrival queue and *B* for a performer being busy or not, as well as (b) three event variables representing *Arrival*, *ProcessingStart* and *ProcessingEnd* events, in the form of circles. In addition, it defines the sequencing of events of those types with the help of *Event Scheduling* arrows, together with caused state changes in the form of (possibly conditional) variable assignments (underneath event circle names).

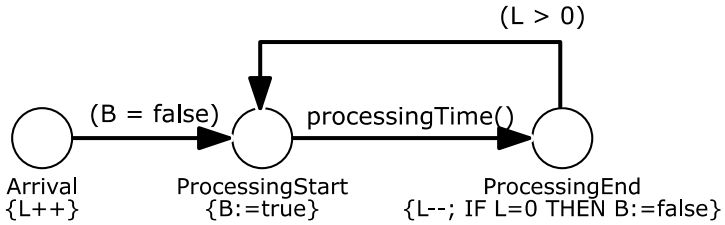


Figure 1 *An Event Graph defining an ES model with two state variables and three event types*

Event Graphs provide a visual modelling language with a precise semantics that captures the fundamental ES paradigm. However, Event Graphs are a rather low-level DES modelling language: they lack a visual notation for (conditional and parallel) branching, do not support OO state structure modelling (with attributes of objects taking the role of state variables) and do not support the concept of activities.

In OEM&S, object types and event types are modeled as special categories of classes («object type» and «event type») in a special kind of UML Class Diagram/Model, called *OE Class Diagram/Model*. *Random variables* are modeled as a special category of class-level operations (designated with «rv») constrained to comply with a specific probability distribution such that they can be implemented as static methods. Finally, *event rules* are modeled in DPMN process diagrams (and possibly also in pseudo-code), such that they can be implemented in the form of special *onEvent* methods of event classes.

In a simulation model, certain types of events are characterized as *exogenous*, while the others are endogenous (or *caused* by previous events). In an OE class model, we therefore categorize event types as either «exogenous event type» or just «event type». The exogenous events of a DES model

correspond to the *Start* events of a BPMN process model, whereas the caused events correspond to BPMN *Intermediate* or *End* events.

In the OE class model shown in Figure 2, *PartArrival*, *ProcessingStart* and *ProcessingEnd* events are associated with a *WorkStation* object (as their only participant). This is the workstation where these events happen. As a class for exogenous events, the *PartArrival* class defines a random variable *recurrence*, which generally determines the recurrence frequency of exogenous events (the elapsed time between two consecutive events of the given type, also called *inter-occurrence time*). The *ProcessingStart* event class defines a random variable *processingTime*.

Object Event (OE) Graphs, as a basic type of DPMN process diagrams, extend the Event Graph diagram language by adding object rectangles containing declarations of typed object variables and state change statements, as well as gateway diamonds for expressing conditional and parallel branching.

A DPMN process model, such as an OE Graph, is based on an underlying information model defining the types of its objects and events. The process model shown in Figure 3 is an OE Graph that is based on the OE class model shown in Figure 2.

Notice that in the OE Graph of Figure 3, the state variables of the Event Graph of Figure 1, *L* and *B*, have been replaced by the attributes *inputBufferLength* and *status* defined in the OE class model of Figure 2.

OE Graphs are a conservative extension of Event Graphs. This means that an OE Graph can be transformed to an Event Graph preserving its dynamics by replacing its objects with corresponding sets of state variables.

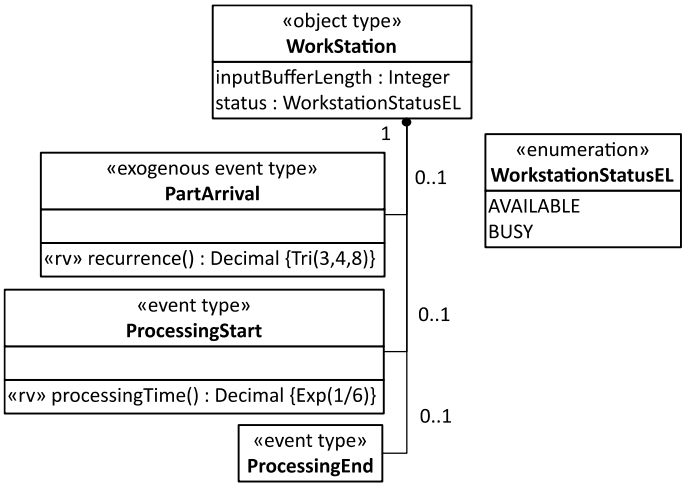


Figure 2 *An OE class model defining an object type and three event types*

Like Petri Nets, OE Graphs have a formal semantics. But while Petri Nets are an abstract computational ("token flow") formalism without an ontological foundation, OE Graphs are based on the ontological categories of objects, events and causal regularities such that an OE Graph can be decomposed into a set of *event rules*, representing causal regularities, which define the transitions of an *Abstract State Machine* [15].

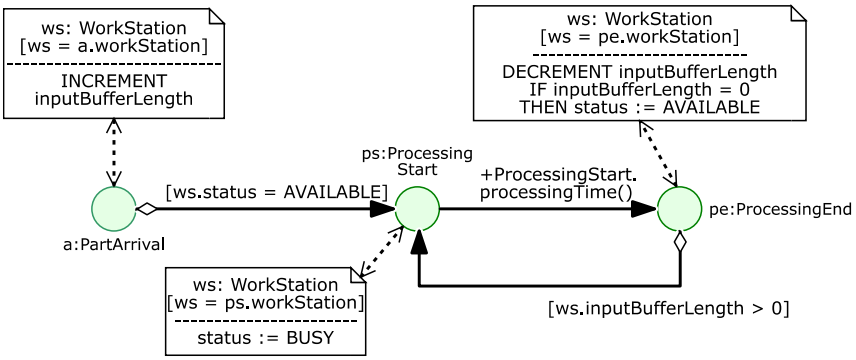


Figure 3 *An Object Event Graph based on the OE class model of Figure 2*

An OE Graph specifies a set of chained event rules, one rule for each event circle of the model. The above OE Graph specifies the following three event rules:

1. On each *PartArrival* event, the *inputBufferLength* attribute of the associated *WorkStation* object is incremented and if the workstation's *status* attribute has the value AVAILABLE, then a new *ProcessingStart* event is scheduled to occur immediately.
2. When a *ProcessingStart* event occurs, the associated *WorkStation* object's *status* attribute is changed to BUSY and a *ProcessingEnd* event is scheduled with a delay provided by invoking the *processingTime* function defined in the *ProcessingStart* event class.
3. When a *ProcessingEnd* event occurs, the *inputBufferLength* attribute of the associated *WorkStation* object is decremented and if the *inputBufferLength* attribute has the value 0, the associated *WorkStation* object's status attribute is changed to AVAILABLE. If the *inputBufferLength* attribute has a value greater than 0, a new *ProcessingStart* event is scheduled to occur immediately.

These event rules can be implemented as *onEvent* methods of their triggering event classes. The resulting model can be run at https://sim4edu.com/oesjs/core1/workstation-1/ as a web-based simulation.

DPMN consists of three layers. The first layer, for modelling OE Graphs, corresponds to an extension of Event Graphs by adding the concept of *Objects*. The second layer (DPMN-A), for modelling Activity Networks, adds the concepts of *Resource-Constrained Activities* and *Resource-Dependent Activity Scheduling* based on resource roles and resource pools, while the third layer (DPMN-PN), for modelling Processing Networks, adds the concepts of *Processing Objects*, *Processing Activities* and *Processing Flows*.

2.2 From Object Event Graphs to Activity Networks

In (Wagner 2020), we have shown how to extend OE Graphs by adding support for resource-constrained activities, resulting in DPMN-A, comprised of three new information modelling elements (*Activity Type*, *Resource Role*, *Resource Pool*) and two new process modelling elements (*Activity* and *Resource-Dependent Activity Scheduling Arrow*). DPMN-A diagrams allow modelling *Activity Networks*.

Conceptually, an activity is a composite event with a non-zero duration that is composed of, and temporally framed by, a pair of instantaneous start and end events. In the transformation shown in Figure 4 below, the pair of *ProcessingStart* and *ProcessingEnd* event types of the OE class model of Figure 2 is replaced with a corresponding *Processing* activity type. This replacement pattern is an essential part of the semantics of activities in DPMN: by reduction to a pair of corresponding start and end events.

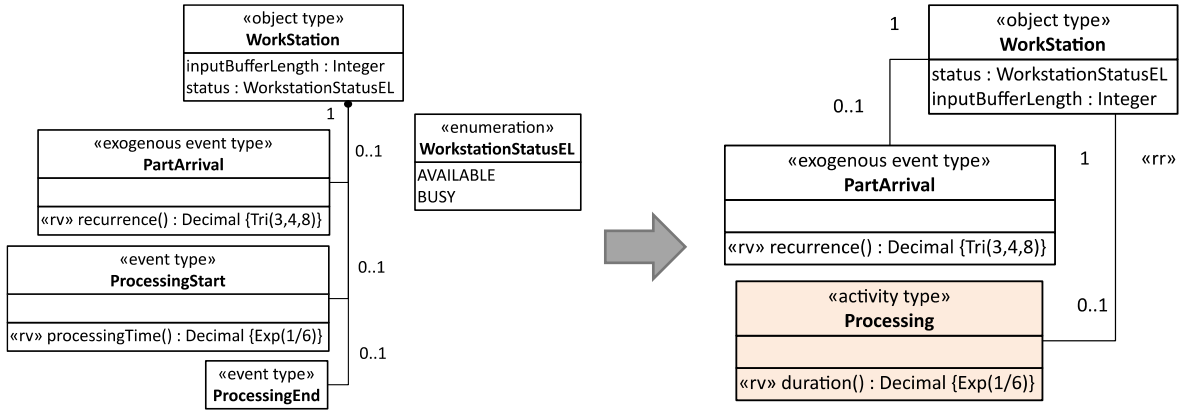


Figure 4 *Rewriting an OE class model with a pair of activity start and end event types to a model with a corresponding activity type (Processing)*

The *Activity-Start-End Rewrite Pattern* exemplified in figures 4 and 5 can also be applied in the inverse direction, replacing an Activity rectangle with a pair of Event circles. It allows reducing an Activity Network model with Activity rectangles to an OE Graph as a basic DPMN diagram without Activity rectangles.

The target model of Figure 5 specifies two event rules:

1. On each *PartArrival* event: (a) if the workstation's *status* is AVAILABLE, then a local rule variable *wsAllocated* is set to *true* and the workstation's *status* is set to BUSY, else the workstation's *inputBufferlength* is incremented; (b) if the *wsAllocated* variable has the value *true*, then a new *Processing* activity is scheduled to start immediately (with a duration provided by invoking the *duration* function defined in the *Processing* activity class).
2. When a *Processing* activity ends: (a) if the workstation's *inputBufferlength* is equal to 0, then the workstation's *status* is set to AVAILABLE, else the local rule variable *wsAllocated* is set to *true* and the *inputBufferlength* is decremented; (b) if the *wsAllocated* variable has the value *true*, a new *Processing* activity is scheduled to start immediately (with a duration provided by invoking the *duration* function defined in the *Processing* activity class).

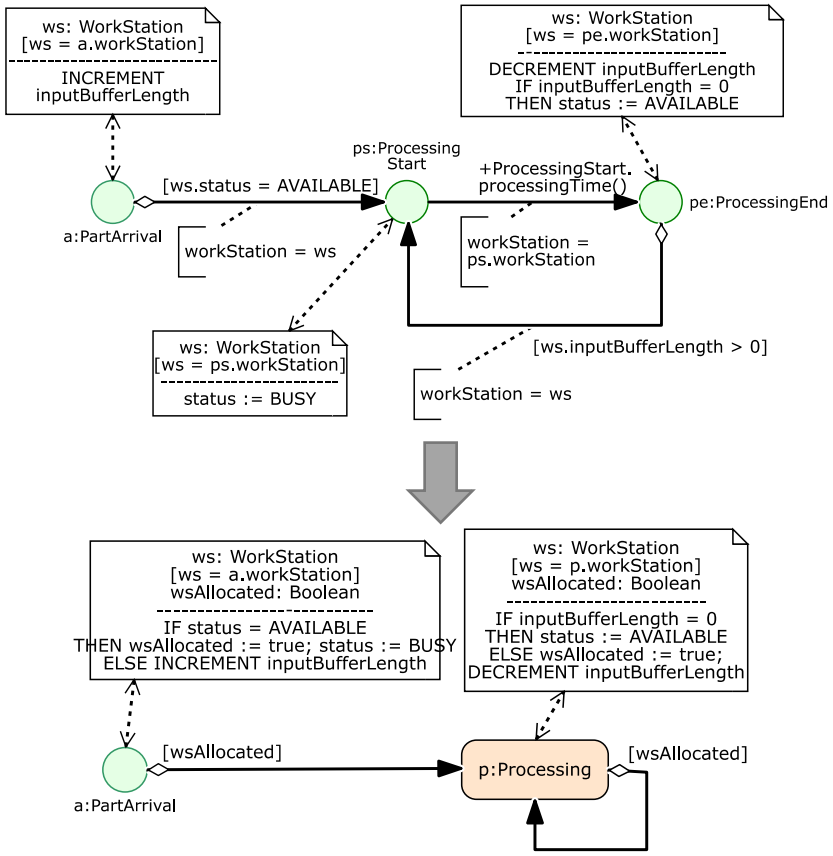


Figure 5 *Rewriting an OE Graph to a corresponding Activity Network model based on the target OE class model of Figure 4*

2.3 From Activity Networks to Processing Networks

The concept of discrete *Processing Networks* is a generalization of the Operations Research concept of *Queueing Networks*. A *Processing Object* enters a processing network via an *Arrival* event at an *Entry Station*, is subsequently routed along a chain of *Processing Stations* where it is subject to *Processing Activities*, and finally exits the network via a *Departure* event at an *Exit Station*. In typical definitions of a queueing network, the processing station is the only required resource of the processing activities

performed at that station, while in a processing network, processing activities can have many required (and optional) resources of various types being allocated with various methods.

PNs have been investigated in *operations management* (Loch 1998) and in the mathematical theory of queuing (Williams 2016), and have been the application focus of most industrial simulation software products, historically starting with GPSS (Gordon 1961) and SIMAN/Arena (Pegden and Davis 1992). They allow modelling many forms of *discrete processing processes* as can be found, for instance, in the manufacturing and services industries.

In the field of DES, PNs have often been characterized by the narrative of "entities flowing through a system". In fact, while in Activity Networks (DPMN-A), there is only a flow of events (including activities), in Processing Networks (DPMN-PN), this flow of events is over-laid with a flow of (processing) objects.

It is remarkable that the PN paradigm has dominated the DES software market since the 1990s and still flourishes today, often with object-oriented and "agent-based" extensions. Its dominance has led many simulation experts to view it as a synonym of DES, which is a conceptual flaw because the concept of DES, even if not precisely defined, is clearly more general than the PN paradigm.

A *Processing Activity* is a resource-constrained activity that is performed at a *processing station* and takes one or more objects as inputs and processes them in some way (possibly transforming them from one type of object to another type), creating one or more objects as outputs. The processed objects have been called "transactions" in GPSS and "entities" in SIMAN/Arena, while they are called *Processing Objects* in DPMN.

Each node in a PN model represents both an object and a typed event variable. An *Entry Node* represents both an *Entry Station* (e.g., a reception area or an entrance to an inventory) and an *Arrival* event variable. A *Processing Node* represents both a *Processing Station* (e.g., a workstation or a room) and a processing activity variable. An *Exit Node* represents both an *Exit Station* and a *Departure* event variable. A *Processing Flow* arrow connecting two processing nodes represents both an event flow and an object flow. Thus, the node types and the flow arrows of a PN are high-level modelling concepts that are overloaded with two meanings.

The (entry, processing and exit) stations of a PN define locations in a network space, which may be abstract or based on a two- or three-dimensional Euclidean geometry. Consequently, PN models are spatial simulation models, while Object Event Graphs and Activity Networks allow to abstract away from space. When a processing object is routed to a follow-up processing station, it moves to the location of that station. The underlying space model allows visualizing a PN simulation in a natural way with processing objects as moving objects.

A PN modelling language should have elements for modelling each of the three types of nodes. Consequently, DPMN-A has to be extended by adding new visual modelling elements for entry, processing and exit nodes, and for connecting them with processing flow arrows.

The simulation modelling concepts of the PN paradigm have been adopted by most DES software products, including Arena, Simio and AnyLogic. However, each of these products uses its own variants of the PN concepts, together with their own proprietary terminology (and proprietary diagram language), as illustrated by Table 1.

Table 1 *Different terminologies used for the same PN modelling concepts*

OEM/DPMN	**Arena**	**Simio**	**AnyLogic**
Processing Object	Entity	Token	Agent
Entry Node	Create	Source	Source
Processing Node	Process	Server	Seize+Delay+Release (or Service)
Exit Node	Dispose	Sink	Sink

For accommodating PN modelling, the Activity Network modelling language OEM-A is extended by adding pre-defined types for processing objects, entry nodes, arrival events, processing nodes, processing activities, exit nodes and departure events, resulting in *OEM-PN*.

As an illustrating example, we re-model the workstation system (considered above) as a Processing Network. Part arrivals are modeled with an «entry node» element (with name "partEntry"), the

workstation is modeled with a «processing node» element, and the departure of parts is modeled with an «exit node» element (with name "partExit").

DPMN is extended by adding the new modelling elements of PN Node rectangles, representing node objects with associated event types, and Processing Flow arrows, representing combined object-event flows. PN Node rectangles take the form of stereotyped UML object rectangles, while PN Flow arrows have a special arrow head, as shown in Figure 6.

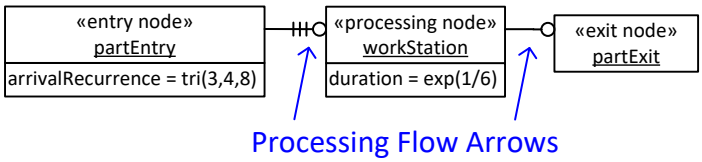


Figure 6 *A PN model of a workstation system using node rectangles and Processing Flow arrows*

The PN model implicitly defines a process model where the workstation process node stands both for a processing activity and a processing station resource object, as shown in the upper part of Figure 7, while using the built-in types shown in the lower part of Figure 7. Since the PN model is completely based on built-in types, no separate OE class model is needed. Notice how the entry node's *arrivalRecurrence* function defines the *recurrence* function of the corresponding *Arrival* event class, and the processing node's *duration* function defines the *duration* function of the corresponding *ProcessingActivity* event class.

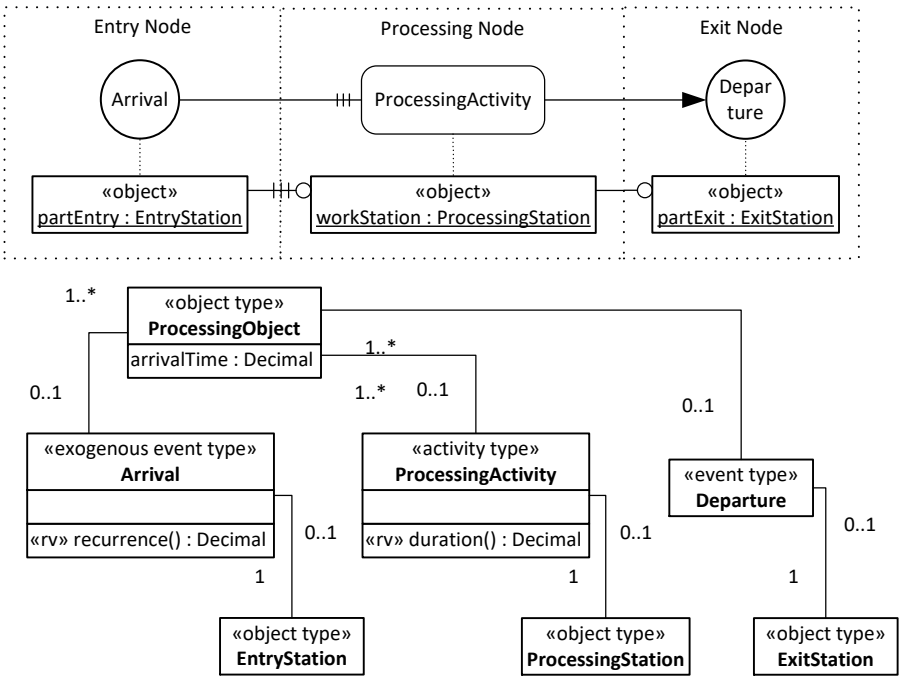


Figure 7 *The elements implicitly defined (or used) by the workstation PN model of Figure 6*

3 CASE STUDY: MAKE AND DELIVER PIZZA

We consider a simple model of a Pizza Service with three consecutive activities: (1) take order, (2) make pizza, and (3) deliver pizza, in a company with 2 order takers, 6 pizza makers, 3 ovens and 10 delivery scooter drivers and scooters. For getting a quick impression, you can run this model from the sim4edu.com website.

For lack of space, we omit discussing the conceptual modelling of this problem and jump directly to the simulation design with DPMN. For a conceptual model, see https://sim4edu.com/reading/des-engineering/make-and-deliver-pizza.

In our simulation design, we make the following simplifications. We consider only one particular pizza service company, which does not have to be modeled as an explicit object. Also, we abstract away from individual customers, orders and pizzas. And we merge the resource roles *delivery scooter driver* and *scooter*, keeping only *scooter*s as resources of *deliver pizza* activities.

We consider a scenario with the following resource pools: 2 order takers, 6 pizza makers, 3 ovens and 10 scooters.

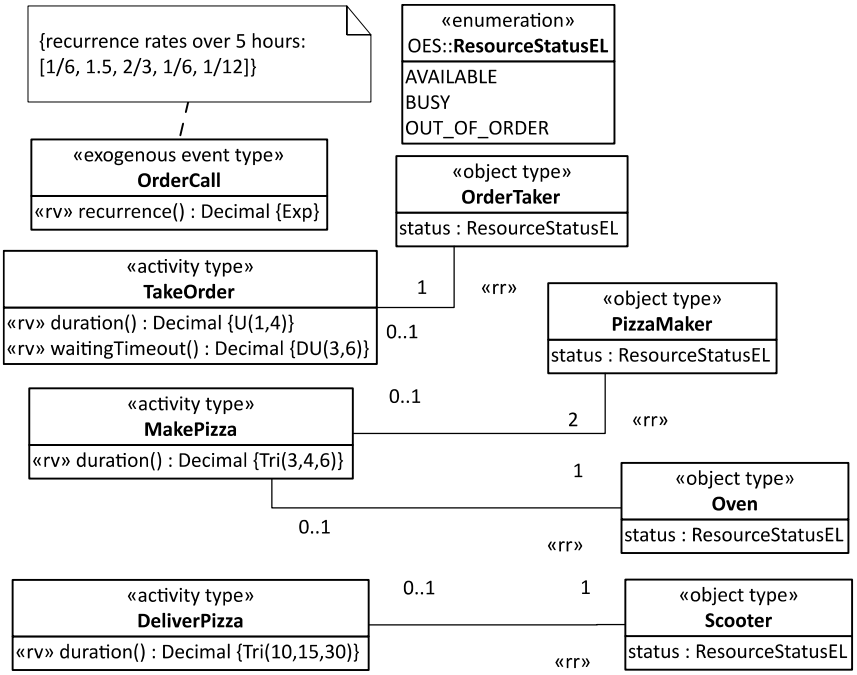


Figure 8 *An OE class model defining object, event and activity types for the Make-and-Deliver-Pizza process model*

Notice that the association end stereotype «rr» stands for "resource role". The OE class model specifies, for instance, that a *MakePizza* activity requires an oven and two pizza makers as resources.

Notice how functions representing random variables, like the duration function of all activity types, are marked with the keyword (or UML 'stereotype') «rv» standing for "random variable". These random variable functions sample from a probability distribution function (PDF), which is symbolically indicated with expressions like *Tri(30,40,50)* standing for the *triangular* PDF with lower and upper bounds 30 and 50 and a median of 40, or *DU(1,4)* standing for the *discrete uniform* PDF with lower and upper bounds 3 and 6.

In the case of the event type *OrderCall*, the random variable function *recurrence* samples from an *exponential* PDF with five different event rates given for the five consecutive hours during which the pizza service operates.

The activity type *TakeOrder* is associated with the object type *OrderTaker* via the implicit resource role *orderTaker* (with a resource cardinality constraint of "exactly 1"), indicated with the association end stereotype «rr» standing for "resource role". A resource role assigns resource objects to activities.

Likewise, *MakePizza* is associated with *PizzaMaker* and *Oven* via the (implicitly named) resource roles *pizzaMakers*, having a resource cardinality constraint of "exactly 2", and *oven*, having a resource cardinality constraint of "exactly 1".

An OE class design diagram like the one above, defines resource roles (like *pizzaMakers*), resource role types (like *PizzaMaker*) and resource cardinality constraints (like "exactly 2") for all types of activities. Normally, in an OE simulation there is a one-to-one correspondence between resource role types and resource pools. By convention, a resource pool has the same name as the corresponding resource role type, yet pluralized and starting with a lowercase character. For instance, the name of the resource pool for *PizzaMaker* is *pizzaMakers*.

Notice that *OrderCall* events are exogenous, having a recurrence function defined case-wise for each of the five hours per day operation of the pizza service company (in the attached invariant box).

For implementing the waiting timeout event defined in the process model, the activity type *TakeOrder* has a class-level *waitingTimeout* function implementing a random variable with PDF *U(3,6)*.

A DPMN process design model (like the one shown in Figure 9) essentially defines the admissible sequences of events and activities (together with their dependencies and effects on participating objects), while its underlying OE class model (like the one shown in Figure 8) defines the types of objects, events and activities, together with the participation of objects in events and activities, including the resource roles of activities, as well as resource cardinality constraints.

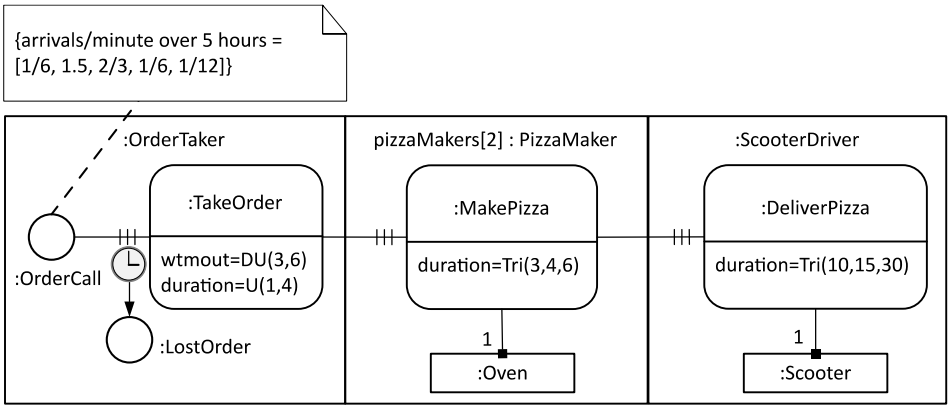


Figure 9 *A process design model for the Make-and-Deliver-Pizza business process*

The process model shown in Figure 9 is enriched by definitions of items from its underlying OE class model, such as activity duration functions or resource dependencies. Such an enriched DPMN process design model includes all computational details needed for an implementation without a separate explicit OE class design model, which is defined implicitly. For instance, the enriched DPMN model of Figure 9 implicitly defines the OE class model of Figure 8 above.

Notice that in the model of Figure 9, performer roles are defined in the form of *Lanes* consisting of a name (such as *pizzaMakers*) and and an object type name (such as *PizzaMaker*) denoting its range, separated by a colon. When the performer role name is appended by a multiplicity expression in brackets, as in *pizzaMakers[2]*, this denotes a resource cardinality constraint (stating that exactly 2 pizzaMakers are required). When only a performer type prefixed with a colon (such as *:OrderTaker*) is provided, this means that the implicit performer role name is obtained by lowercasing the performer type name (as in *orderTaker:OrderTaker*).

Notice that the model of Figure 9 does not include any element representing a resource pool. It is assumed that for any organizational position described in the underlying OE class model, 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.

In the following subsections, we show how to implement the Make-and-Deliver-Pizza business process model shown in Figure 9 with AnyLogic and Simio. In addition to the *simulation model* expressed by this DPMN process diagram, we also need the data of a *simulation scenario*, like the initial states of variables and objects, including the resource pools, for being able to run a simulation. We use a baseline scenario with 2 order takers, 6 pizza makers, 3 ovens, and 10 scooters.

Since Simio and AnyLogic do not support Activity Networks, but only Processing Networks (PN) with "entities flowing through the system", we need to impose a PN view on the Make-and-Deliver-Pizza business process. This requires to figure out what could be used as "entities" for being able to make a PN model.

Since in the real pizza service system there are no entities that flow through the system, we need to assume an artificial abstract entity like "the order", which arrives at the order taker and then takes the form of the ordered pizza being delivered to the customer.

3.1 Implementation with AnyLogic

An enriched DPMN process design model, like the Make-and-Deliver-Pizza process model above, can be implemented with AnyLogic's *Process Modelling Library* by taking the following steps:

1. An exogenous event circle, like *OrderCall*, is turned into an entry node *OrderCall* (an AnyLogic "Source" element). Since the *OrderCall* event recurrence is defined case-wise for each of the five hours of the pizza service's operation, we set the field *Arrivals defined by* to "Rate schedule" and set the field *Rate schedule* to the schedule that defines the arrivals ("ArrivalSchedule").
2. The *TakeOrder* activity rectangle is implemented as a corresponding processing node (an AnyLogic "Service" element). Its performer role *orderTaker:OrderTaker* (in the diagram above abbreviated by *:OrderTaker*) is turned into an AnyLogic resource pool "orderTakers" (with its *Capacity* field set to 2). The "Service" element's *Seize* field is set to "units of the same pool" and its *Resource pool* field is set to the previously defined pool "orderTakers" (by selecting it from the drop-down list). The field *Delay time* is set to an AnyLogic Java expression corresponding to the activity's duration value: *uniform(1,4)*.
3. Likewise, the *MakePizza* activity rectangle is implemented as a corresponding "Service" element with two resource pools "pizzaMakers" and "ovens" modeled after the corresponding performer roles (specifying a resource multipliclity of "2" for "ovens" in the field *Resource sets*). The field *Delay time* is set to *triangular(3,6,4)*.
4. Likewise, the *DeliverPizza* activity rectangle is implemented as a corresponding "Service" element in a similar way as *MakePizza*.
5. Finally, the AnyLogic model is completed by appending a "Sink" element (called *DeliveredOrder* in the AnyLogic process diagram below) to the *DeliverPizza* "Service" element (representing the process end activity).

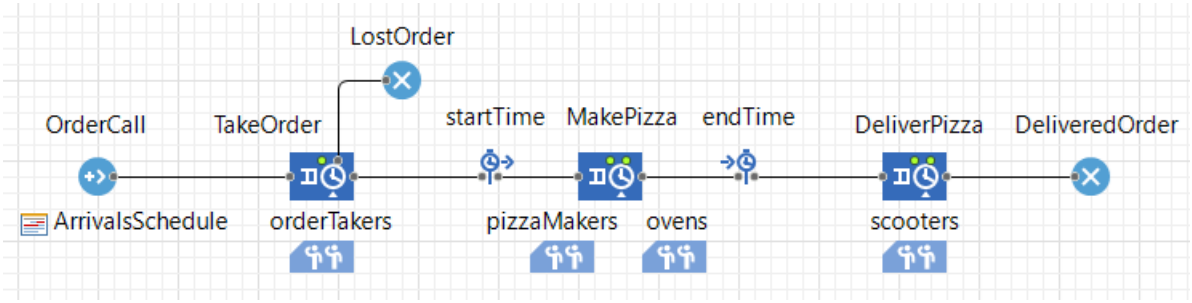


Figure 10 *An AnyLogic process diagram for the Make-and-Deliver-Pizza business process*

3.2 Implementation with Simio

For implementing the Make-and-Deliver-Pizza model with Simio, we use Simio's *Facility* view and its Standard Library where the PN modelling concept of an *entry node* is called "Source", a *processing node* is called "Server" and an *exit node* is called "Sink". We (1) first drag and drop a "Source" element from Simio's Standard Library and rename it to *OrderCall*, followed by (2) three "Server" elements, renamed to *TakeOrder*, *MakePizza* and *DeliverPizza*, followed by (3) a "Sink" element renamed to *ReceivePizza*.

In addition, we define:

1. two individual Resource objects *orderTaker1* and *orderTaker2*, which are placed in a Simio Object List *orderTakers* representing a resource pool for the *TakeOrder* activity;

2. six individual Resource objects *pizzaMaker1*, ..., *pizzaMaker6*, which are placed in a Simio Object List *pizzaMakers* representing a resource pool for the *MakePizza* activity*;*
3. two count pools *ovens* and *scooters* in the form of Simio Resource objects with capacities 3 and 10, respectively.

This results in the following Simio process diagram:

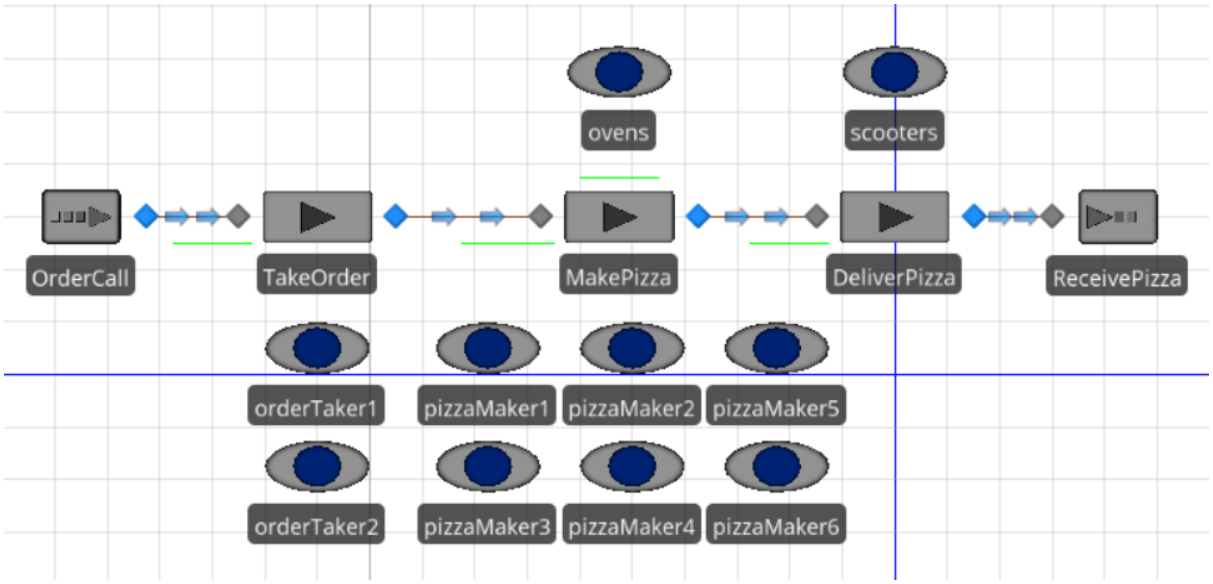


Figure 11 *A Simio process diagram for the Make-and-Deliver-Pizza business process*

REFERENCES

Banks J, Carson J S, Nelson B L, and Nicol D M (2010). *Discrete-Event System Simulation*. 5th ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Banks J (1998). Principles of Simulation. In: Banks J (eds). *Handbook of Simulation*. New York: John Wiley & Sons, Inc.

BPMN (Version 2.0), 2011. http://www.omg.org/spec/BPMN/2.0, accessed 5th July 2021.

Gordon, G. 1961. "A general purpose systems simulation program". In *AFIPS '61: Proceedings of the Eastern Joint Computer Conference*, 87–104, New York: Association for Computing Machinery.

Guizzardi, G., and G. Wagner. 2010. "Towards an Ontological Foundation of Discrete Event Simulation". In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, E. Yücesan. 652−664. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gurevich, Y. 1985. "A New Thesis". *Abstracts, American Mathematical Society*, 6(4):317.

Loch, C.H. 1998. Operations Management and Reengineering. European Management Journal, 16, 306−317.

Markowitz, H., B. Hausner, and H. Karr. 1962. *SIMSCRIPT: A Simulation Programming Language*. Englewood Cliffs, N. J.: Prentice Hall.

Pegden, C.D. and D.A. Davis. 1992. "Arena: a SIMAN/Cinema-Based Hierarchical Modelling System". In *Proceedings of the 1992 Winter Simulation Conference*, edited by J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, 390–399. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pegden, C.D. 2010. "Advanced Tutorial: Overview of Simulation World Views". In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 643−651. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schruben, L.W. 1983. “Simulation Modelling with Event Graphs”. *Communications of the ACM* 26:957–963.

Tocher, K.D. 1960. An Integrated Project for the Design and Appraisal of Mechanized Decision-Making Control Systems. Operational Research 11(1/2) :50–65.

Wagner, G. 2022. *Discrete Event Simulation Engineering*. https://sim4edu.com/reading/des-engineering/, accessed 1 December 2022.

Wagner, G. 2021. “Business Process Modelling and Simulation with DPMN: Processing Activities”. In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo and M. Loper. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wagner, G. 2020. “Business Process Modelling and Simulation with DPMN: Resource-Constrained Activities”. In *Proceedings of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing. 45–59. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wagner, G. 2018. “Information and Process Modelling for Simulation – Part I: Objects and Events”. *Journal of Simulation Engineering* 1:1–25.

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. 762–773. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Williams, R.J. 2016. Stochastic Processing Networks. *Annual Review of Statistics and Its Application* 3:1, 323–345.

AUTHOR BIOGRAPHY

GERD WAGNER is Professor of Internet Technology in the Dept. of Informatics, Brandenburg University of Technology, Germany. After studying Mathematics, Philosophy and Informatics in Heidelberg, San Francisco and Berlin, he (1) investigated the semantics of negation in knowledge representation formalisms, (2) developed concepts and techniques for agent-oriented modelling and simulation, (3) participated in the development of a foundational ontology for conceptual modelling, the *Unified Foundational Ontology (UFO)*, and (4) created a new Discrete Event Simulation paradigm, *Object Event Modelling and Simulation (OEM&S)*, and a new process modelling language, the *Discrete Event Process Modelling Notation (DPMN)*. Much of his recent work on OEM&S and DPMN is available from sim4edu.com and dpmn.info.