Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

BUSINESS PROCESS MODELING AND SIMULATION WITH DPMN: RESOURCE-CONSTRAINED ACTIVITIES

Gerd Wagner

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

ABSTRACT

This tutorial article, which is extracted from (Wagner 2019), shows how to use UML Class Diagrams and Discrete Event Process Modeling Notation (DPMN) Process Diagrams for making simulation models of business processes with resource-constrained activities based on the DES paradigm of Object Event Modeling and Simulation. In this approach, the state structure of a business system is captured by a UML Class Diagram, which defines the types of objects, events and activities underlying a DPMN Process Diagram, which captures the causal regularities of the system in the form of a set of event rules. DPMN Process Diagrams extend the Event Graphs proposed by Schruben (1983) by adding elements from the Business Process Modeling Notation (BPMN), viz. data objects and activities, and, as its main innovation over BPMN, resource-dependent activity start arrows.

1 INTRODUCTION

Object Event (OE) Modeling and Simulation (M&S) is a new general Discrete Event Simulation (DES) paradigm proposed by Wagner (2018), combining *object-oriented* modeling and *event-based* simulation (with *event scheduling*). OEM&S is based on the idea that both conceptual models for DES and DES design models consist of (1) an *information model* and (2) a *process model*. In the case of *conceptual modeling*, a conceptual information model describes the types of objects and events representing the main entities of the real-world system under investigation, while a conceptual process model describes its dynamics in the form of a set of conceptual *event rule models* that capture the *causal regularities* of the system.

In the case of *simulation design modeling*, an information design model prescribes (defines) the types of all objects and events that are relevant for the purpose of a simulation study, thus defining the state structure of a DES system, while a process design model defines the dynamics of a DES system by defining, for all event types defined by the underlying information design model, an *event rule design model* that specifies the state changes and follow-up events implied by the occurrence of an event of that type.

In (Wagner 2018), we have introduced a variant of the *Business Process Modeling Notation (BPMN)*, called *Discrete Event Process Modeling Notation (DPMN)*, and have shown how to use UML Class Diagrams and DPMN Process Diagrams for making basic OE models 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* in the sense of Gurevich (1985). This fundamental characterization of an OE model provides a formal (operational) semantics for OE Simulation (OES) by defining an *OES formalism* that any OE simulator has to implement.

In this tutorial article, we show how to extend basic OEM/DPMN for adding support for *activities*, resulting in an extension, OEM/DPMN-A, comprising four new information modeling elements (Activity

Type, Resource Role, Resource Pool, and Resource Type) and two new process modeling elements (Activity and Resource-Dependent Activity Start Arrow).

2 BASIC OEM&S

2.1 Ontological Considerations

Ontologically, an **activity** is a composite event (composed of at least a start and an end event) with a duration greater than zero, performed by an agent (a human or another living being, a robot or another artificial agent, or an organization or another social agent). As opposed to activities, activity start and end events are *instantaneous* (zero-duration) events.

As an event, an activity has objects that *participate* in it. In the real world, an activity has at least one participant: the *performer* of the activity. Consequently, a conceptual model should, for each activity type, include the type of objects that play the performer role for activities of that type.

However, in a simulation design model we may leave the performer of an activity implicit and model an activity without modeling any participant. Consequently, a basic *OE* simulator, the core classes of which are described in Figure 1, does not need to support the distinction between objects and agents.

A **discrete process** (instance) consists of a partially ordered set of events that happen in a coherent spatio-temporal region determined by the events' participants and the causal regularities involved. When two or more events within a process have the same order rank, this means that they occur simultaneously.

There are many examples of discrete processes in various domains: (1) in biology, the population dynamics of one or more species living in a certain ecosystem (such as the well-known predator-prey model); (2) in sociology, the process of gossip spreading among a community; (3) in economics, a market based on offers and transactions.

A **business process** (instance) is a discrete process that happens in the context of an *organization*. Typically, a business process is an instance of a business process type defined by an organization (or organizational unit), which is the owner of the business process type, in the form of a process model. Notice that this concept includes business system processes, where many business actors perform activities for handling many business cases in parallel. Consequently, it is more general than the common concept of a business process as a case-handling process, which prevails in the *Information Systems* field of *Business Process Management*.

2.2 Object Event Simulation

The Object Event Simulation (OES) paradigm is based on the idea of executing an OE model starting with an initial simulation state by successively applying the event rules of the model to the evolving simulation states. Figure 1 depicts the core classes of individuals an OE simulator has to deal with at runtime.

Notice that the *occurrence time* of an activity is the time when it completes, that is, it is equal to *startTime + duration*. Typically, the duration of an activity in a simulation run is known, and set, when it is started. An activity type is normally defined with a fixed duration or a random variable duration for all activities of that type. This allows a simulator to schedule the activity's end event when the activity is started. However, in certain cases, an activity type may not define a preset duration, but leave the duration of activities of that type open. When such an activity is still ongoing, it does only have a start time, but no duration and no occurrence time.

2.3 Illustrating Basic OEM Concepts with an Example

As an example of basic OEM&S, we present a simple OE model of a manufacturing workstation that receives parts and stores them in its input buffer for processing them successively. Such a model consists of (1) a *conceptual model* describing the real-world domain, and (2) a *simulation design model* prescribing a certain computational solution for the purpose of a simulation study. Both conceptual models and design models consist of an *information model* describing/defining the system's *state structure* and a *process model*

describing/defining the system's *dynamics*. An information design model defines the object and event types as the basis of a corresponding process design model.

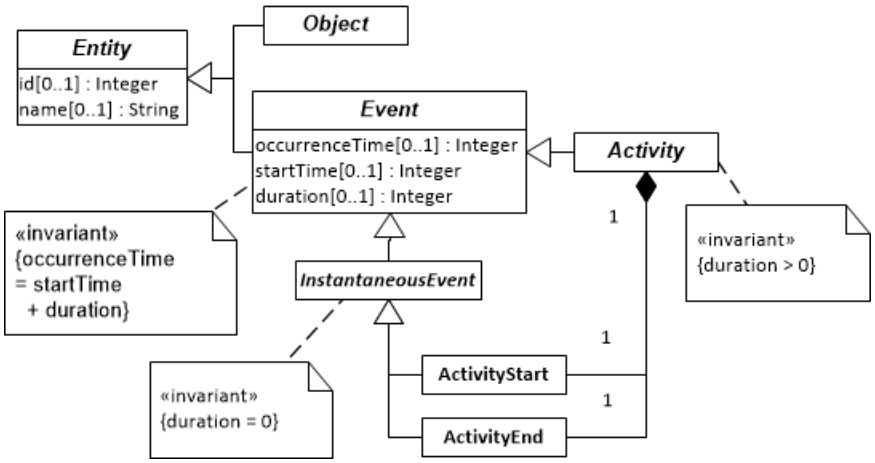


Figure 1: The core classes of individuals an OE simulator has to deal with at runtime.

2.3.1 Conceptual Model

A conceptual information model of a workstation system, defining two object types and four event types, is shown in Figure 2.

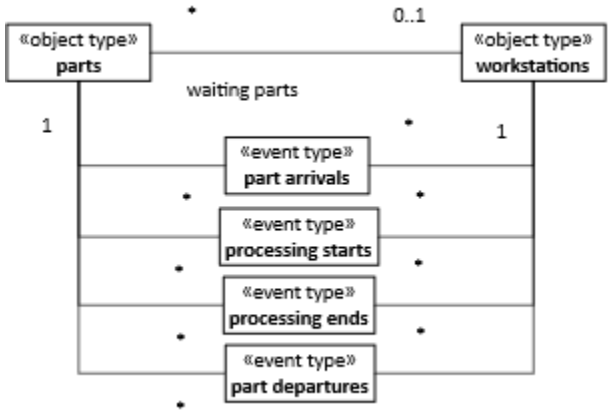


Figure 2: A conceptual information model of manufacturing workstation systems.

As expressed by the associations between the four event types and the two object types, for all four types of events, there are the same two types of objects participating in them: parts and workstations, implying that each event of these four types involves a specific part and a specific workstation.

Notice that the input buffer (filled with waiting parts) is modeled as an association end with name *waiting parts* at the *parts* side of the association between *parts* and *workstations*, expressing the fact that at any point in time, a workstation has zero or more parts waiting in its input buffer for being processed.

A conceptual process model of this system, describing four *causal regularities* in the form of *event rules*, one for each type of event, is shown in Figure 3 in the form of a *BPMN Process Diagram* using Event circles connected with Sequence Flow arrows expressing (conditional) causation, and Data Objects attached to Event circles.

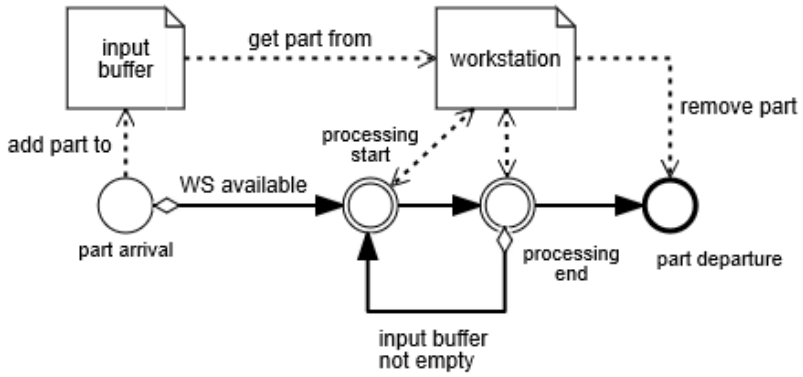


Figure 3: A conceptual process model of a manufacturing workstation system.

The four event rules described by the model shown in Figure 3 are

1. When a part arrives, it is added to the input buffer and, if the workstation is available, there will be a processing start event for processing the newly arrived part.
2. When a processing start event occurs, the next part from the input buffer is being processed and a processing end event is caused to occur sometime later (after the processing time has elapsed).
3. When a processing end event occurs, this will cause a part departure event and, if the input buffer is not empty, another processing start event involving the next part from the buffer.
4. When a part departure event occurs, the processed part will be removed from the workstation.

While BPMN requires to categorize all Event circles into one of the three categories Start or Intermediate or End Event and use a different visual syntax for them, this is not the case in DPMN.

2.3.2 Design Model

A simulation design model is based on a conceptual model. Depending on the purposes/goals of a simulation study, it may abstract away from certain elements of the real-world domain described by the conceptual model, and it adds computational elements representing design decisions, such as *random variables* expressed in the form of random variate sampling functions based on specific probability distributions for modeling the random variation of certain system variables.

An information design model of the single workstation system described above is shown in Figure 4. This model defines the multi-valued `waitingParts` association end to be ordered, which means that it corresponds to a multi-valued reference property holding an ordered collection (such as an array list or a queue) as its value.

The information design model of Figure 4 defines that a *PartArrival* event must reference both a *Part* and a *WorkStation*, representing situations where specific parts arrive at specific workstations. Notice that, computationally, this model requires creating new *Part* objects (or retrieving them from an object pool) before a new *PartArrival* event is created (or scheduled), while it is more common in simulation models to create a new *Part* object only when an arrival event has occurred, which can be modeled by defining a multiplicity of 0..1 for the *Part* end of the *PartArrival-Part* association (with the meaning that *PartArrival* has an optional, instead of a mandatory, reference property with name *part*).

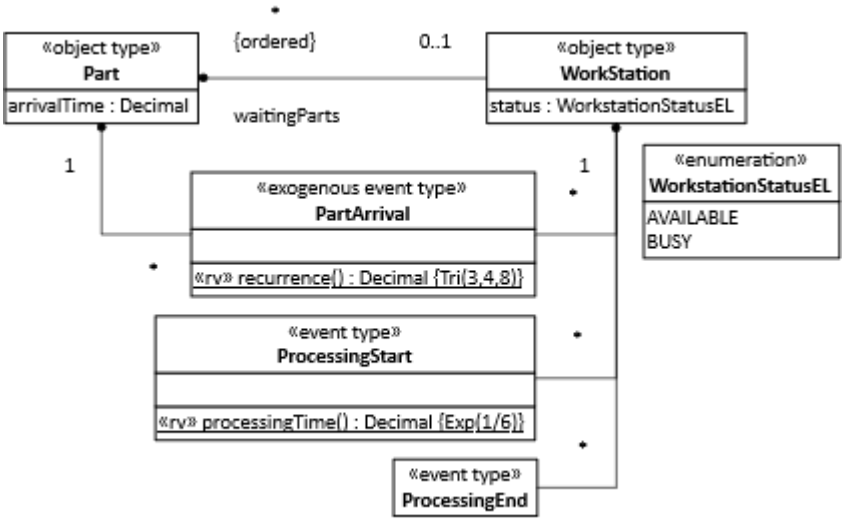


Figure 4: An information design model.

Notice that the model defines two class level operations (designated with the stereotype «rv») implementing random variate sampling functions: `PartArrival::recurrence()` complies with a triangular probability distribution with minimum, mode and maximum parameter values 3, 4 and 8, while `ProcessingStart::processingTime()` complies with an exponential distribution with an event rate parameter value of 6.

A process design model based on the object and event types defined by the information design model of Figure 4 and derived from the conceptual process model of Figure 3 is shown in Figure 5.

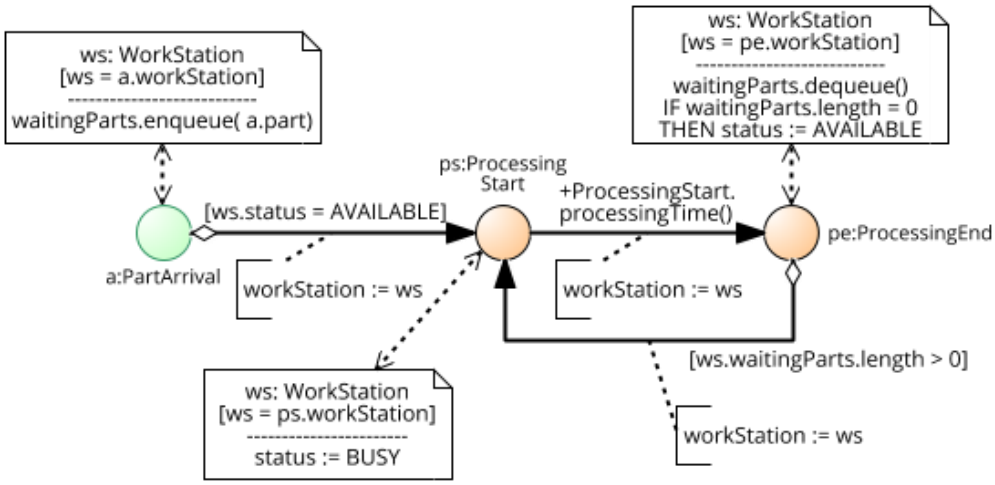


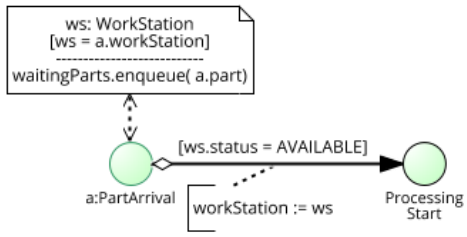
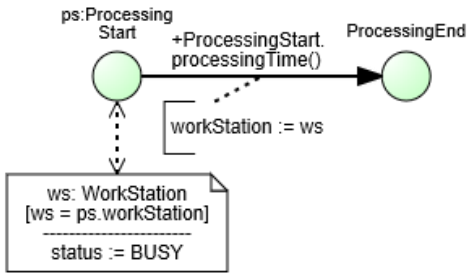
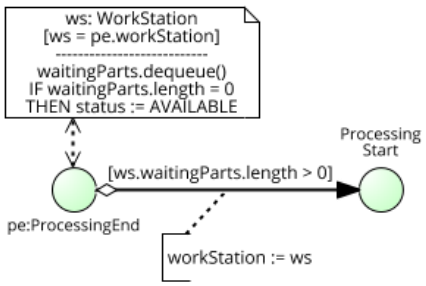
Figure 5: A process design model in the form of a DPMN Process Diagram.

Notice that, since all events happen at the same workstation, all three event scheduling arrows are annotated with the same event property assignment workStation := ws, which simply propagates the object reference to the given workstation along the event scheduling chain. Such property propagation assignments (in event property assignment annotations), where a property value of a follow-up event is set to the corresponding property value of the scheduling (or triggering) event, will be omitted (as implied by event types having the same property names) for avoiding to clutter the process model diagrams.

A DPMN Process Diagram, like the one shown in Figure 5, can be split up into a set of event rule diagrams, one for each of its Event circles, as shown in Table 1. This reduction of a DPMN process design model to a set of event rule design models, together with the operational semantics of event rules presented in (Wagner 2017), provides the semantics of DPMN Process Diagrams.

Notice that an event rule design model can also be expressed textually in the form of a pseudo-code block with four parts: part 1 indicates the triggering event type and declares a rule variable representing the triggering event, part 2 declares further rule variables and initializes them, part 3 contains a state change script consisting of state change statements, and part 4 schedules follow-up events.

Table 1: Event rule design models.

Rule design model	**Pseudo-code**
	**ON a:PartArrival** ws : WorkStation ws := a.workStation ws.waitingParts.enqueue(a.part) IF ws.status = AVAILABLE THEN SCHEDULE ProcessingStart(workStation:=ws)
	**ON ps:ProcessingStart** ws : WorkStation ws := ps.workStation ws.status := BUSY SCHEDULE ProcessingEnd(workStation:=ws) DELAYED BY ProcessingStart.processingTime()
	**ON pe:ProcessingEnd** ws : WorkStation ws := pe.workStation ws.waitingParts.dequeue() IF ws.waitingParts.length = 0 THEN ws.status := AVAILABLE IF ws.waitingParts.length > 0 THEN SCHEDULE ProcessingStart(workStation:=ws)

3 SIMPLE ACTIVITIES

A simple activity is an activity with zero or more participants, none of which is having a special meaning (such as being a resource or a processing object).

3.1 Conceptual Modeling of Simple Activities

Conceptually, an activity is a composite event that is temporally framed by a pair of start and end events. Consequently, whenever a model contains a pair of related start and end event types, like *processing start* and *processing end* in the model of a manufacturing workstation shown on the left-hand side of Figure 6 and Figure 7, they can be replaced with a corresponding activity type, like *processing*, as shown on the right-hand side.

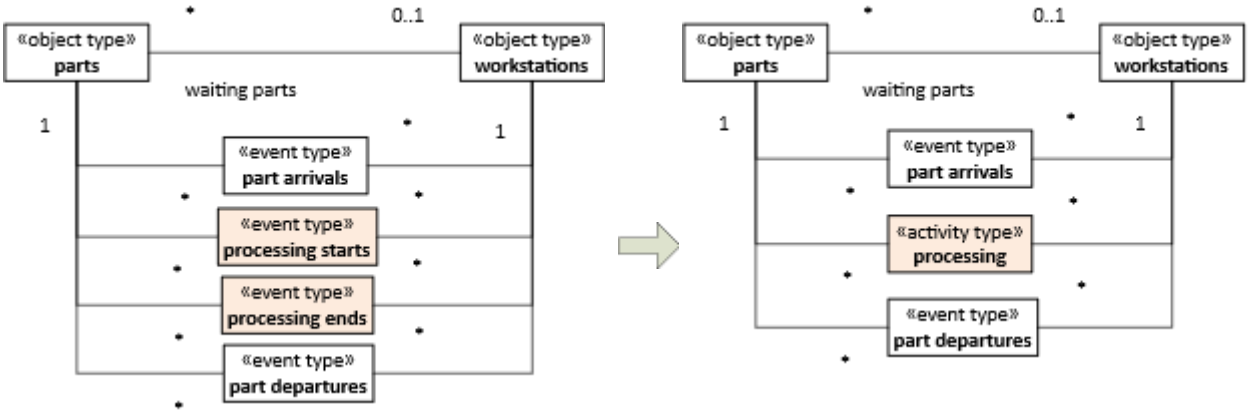


Figure 6: Introducing an activity type in a conceptual information model.

It is obvious that applying this replacement pattern leads to a conceptual and visual simplification of the models concerned.

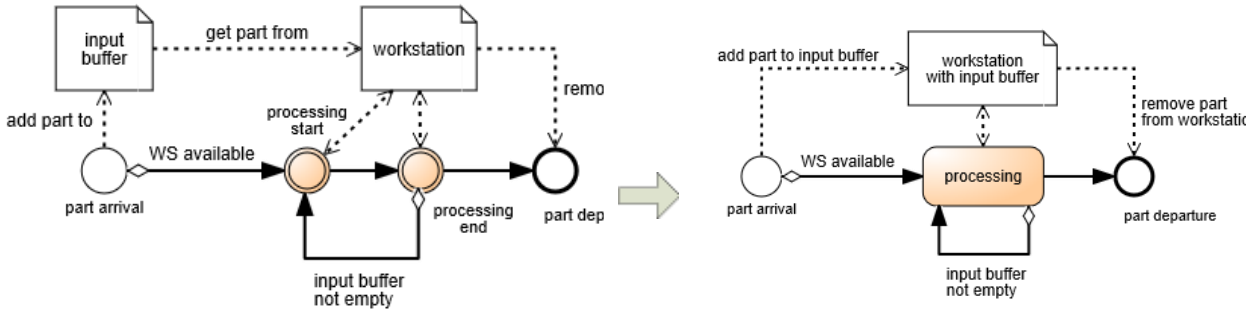


Figure 7: Introducing an activity type in a conceptual process model.

3.2 Design Modeling of Simple Activities

Like in a conceptual model, also in a design model, a pair of corresponding activity start and end event types (or Event circles), like `ProcessingStart` and `ProcessingEnd` in the source models shown in Figure 8 and Figure 9, can be replaced with a corresponding activity type (or Activity rectangles), like `Processing`, as in the target models shown in these figures.

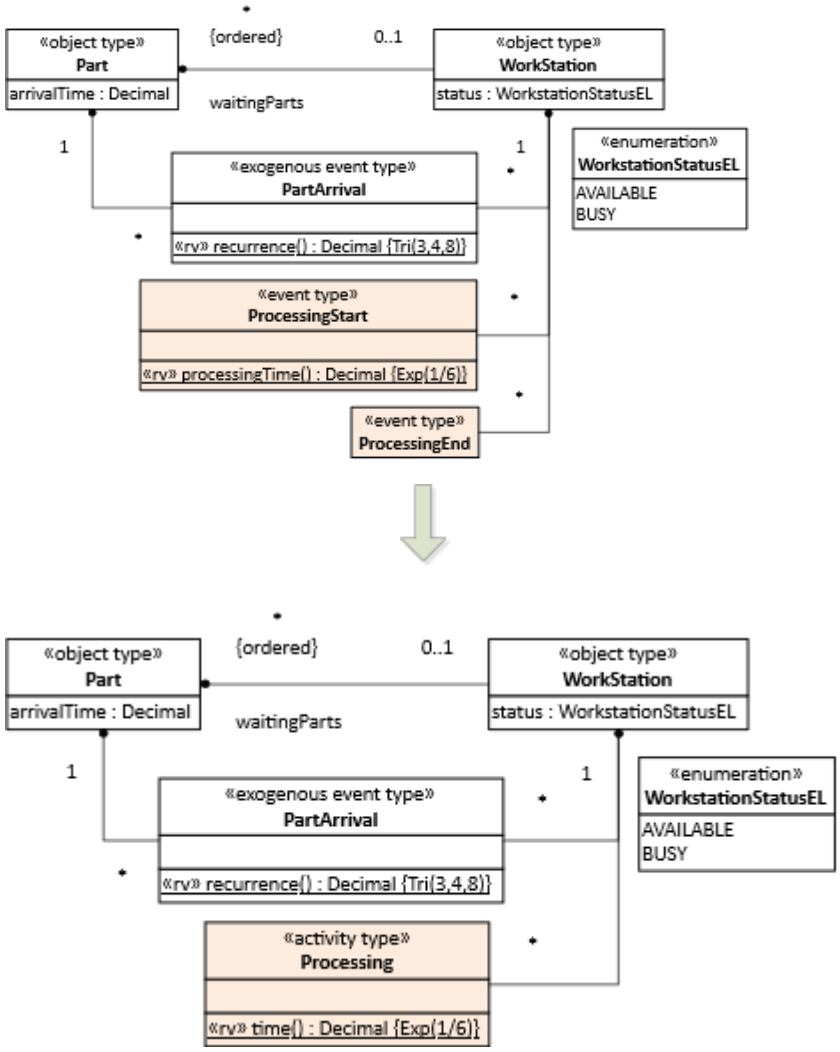


Figure 8: Extending basic OEM to OEM-A class models by introducing activity types.

In the case of an information design model, this replacement pattern implies allocating all features (attributes, associations and operations) of the classes defining the start and the end event type in the class defining the corresponding activity type, possibly with renaming some of them. In the example of Figure 7, there is only one such feature: the class-level operation `ProcessingStart::processingTime`, which is allocated to `Processing` and renamed to `time`.

In the case of a process design model, the replacement pattern implies that an Event circle pair consisting of an Event circle intended to represent an activity start event type and an Event circle intended to represent an activity end event type, with an event scheduling arrow from the activity start to the activity end event circle annotated by a delay expression, is replaced by an Activity rectangle such that:

1. All Data Objects attached to the activity end event circle get attached to the Activity rectangle (since an activity occurs when it is completed).

2. All event scheduling arrows going out from the activity end event circle are turned into event scheduling arrows going out from the Activity rectangle.
3. All activity start event scheduling arrows are replaced with corresponding activity scheduling arrows having an additional creation parameter assignment for the *duration* of a scheduled activity, which is set to the delay expression defined for the activity end event scheduling arrow. In the example above, `Processing::time()` in the target diagram is the same as the delay `ProcessingStart::processingTime` in the source diagram.
4. When the activity start event circle has one or more attached Data Objects or any outgoing event scheduling arrow that does not go to the activity end event circle, then an activity start event circle has to be included in the Activity rectangle for attaching the Data Object(s) and as the source of the outgoing event scheduling arrow(s).

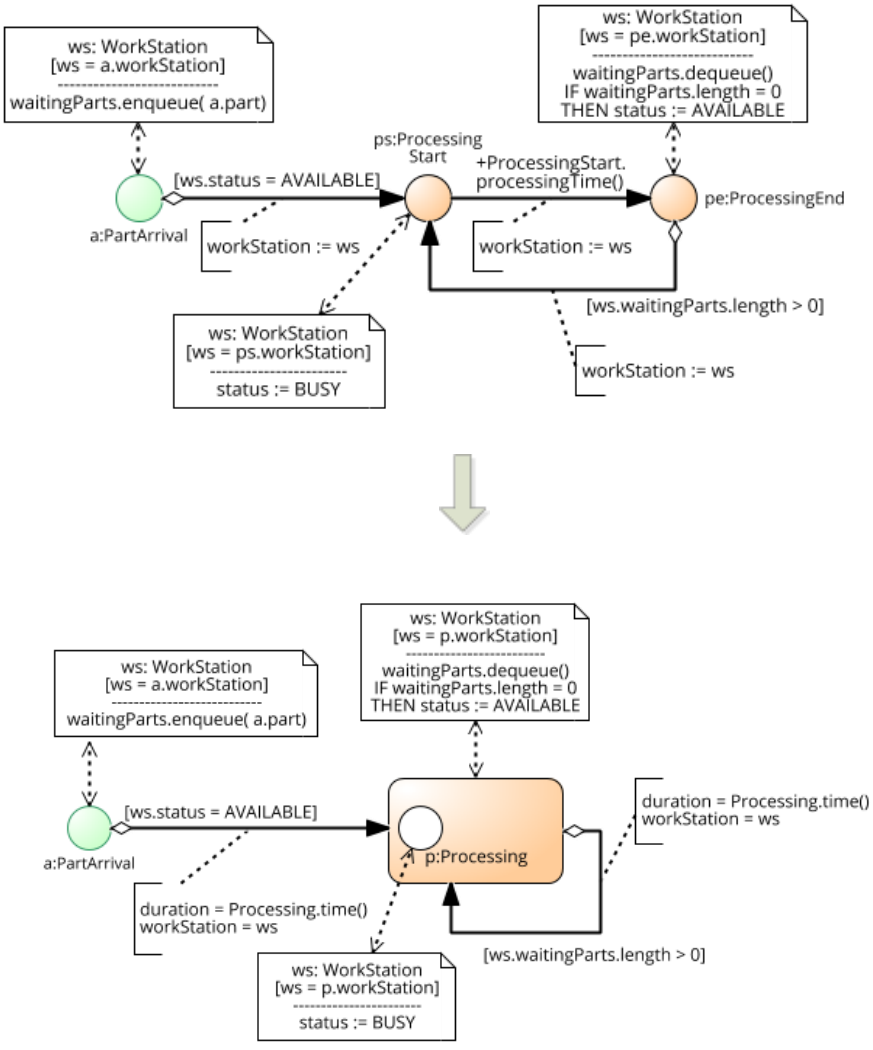


Figure 9: Extending basic DPMN to DPMN-A process models by introducing Activity rectangles.

This *Activity-Start-End Rewrite Pattern*, which can also be applied in the inverse direction, replacing an Activity rectangle with an Event circle pair, defines the meaning of an Activity rectangle in a DPMN

diagram. It allows reducing a DPMN-A diagram with Activity rectangles to a basic DPMN diagram without Activity rectangles.

Notice that the target model of Figure 9 specifies two event rules:

1. On each *part arrival*, if the workstation's *status* is AVAILABLE, then the rule variable *wsAllocated* is set to *true* and the workstation's *status* attribute is set to BUSY, else the arrived part is added to the workstation's input buffer *waitingParts*. If the rule variable *wsAllocated* has the value *true*, then a new *Processing* activity is scheduled to start immediately with its (inherited) *duration* attribute set to the value obtained by invoking the *time* function defined in the *Processing* activity class.
2. When a *Processing* activity ends, if the workstation's input buffer *waitingParts* is empty, then the workstation's *status* attribute is set to AVAILABLE, else the rule variable *wsReallocated* is set to *true* and the next part is removed from the input buffer *waitingParts*. If the rule variable *wsReallocated* has the value *true*, then a new *Processing* activity is scheduled to start immediately with its (inherited) *duration* attribute set to the value obtained by invoking the *time* function defined in the *Processing* activity class.

Notice that a workstation is an exclusive resource of its processing activity. The concepts of resources and resource-constrained activities are discussed in the following sections.

4 RESOURCE-CONSTRAINED ACTIVITIES

A *Resource-Constrained Activity* is an activity where one or more participants play a *Resource Role* (such as *Performer*). Typically, a Resource-Constrained Activity is a component of a business process that happens in the context of an organization or organizational unit, which is associated with the activity as its *Process Owner*.

An activity of a certain type may require certain resources for being performable. At any point in time, a resource required for performing an activity may be *available* or not. A resource is not available, for instance, when it is *busy* or when it is *out of order*.

Resources are objects of a certain type. The resource objects of an activity include its *performer*. While in a conceptual model, describing a real-world system, a performer is required for any activity, a simulation design model may abstract away from the performer of an activity.

For instance, a consultation activity may require a consultant and a room. Such *resource constraints* are defined at the type level. When defining the activity type *Consultation*, these resource constraints are defined in the form of two mandatory associations with the object types *Consultant* and *Room* such that both associations' ends have the multiplicity 1 (“exactly one”). Then, in a simulation run, a new *Consultation* activity can only be started, when both a *Consultant* object and a *Room* object are available.

For all resource-constrained activities, a simulator can automatically collect the following statistics:

1. For each activity type,
    1. the (average, maximum, etc.) *queue length* of its queue of planned activities;
    2. the (average, maximum, etc.) *cycle time*, which is the sum of the waiting time and the activity duration;
    3. the percentage of time each involved resource object is busy with an activity of that type (its *utilization* by activities of that type).
2. The percentage of time each resource object is idle or out-of-order.

For modeling resource-constrained activities, we need to define their types. A resource-constrained activity type is composed of

1. a set of *properties* and a set of *operations*, as any entity type,

2. a set of **resource roles**, each one having the form of a reference property with a name, an object type as range, and a multiplicity that may define a **resource constraint** like, e.g., "exactly one resource object of this type is required" or "at least two resource objects of this type are required".

The resource roles defined for an activity type may include the performer role. A simulation language for simulating activities needs to allow defining activity types with two kinds of properties: ordinary properties and resource roles. At least for the latter ones, it must be possible to define multiplicities for defining resource constraints. These requirements are fulfilled by OEM Class Diagrams where resource roles are defined as stereotyped properties using the stereotype «resource role» or, shorter, «res».

The extension of basic OEM by adding the concepts needed for modeling resource-constrained activities (in particular, resource roles with constraints, resource pools, and resource-dependent activity start arrows) is called *OEM-A*.

4.1 Conceptual Modeling of Resource-Constrained Activities

Modeling resource-constrained activities has been a major issue in the field of Discrete Event Simulation (DES) since its inception in the nineteen-sixties, while it has been neglected and is still considered an advanced topic in the field of Business Process Modeling (BPM). For instance, while BPMN allows assigning resources to activities, it does not allow modeling resource pools, and does neither allow specifying resource cardinality constraints nor parallel participation multiplicity constraints.

In the DES paradigm of *Processing Networks*, Gordon (1961) has introduced the resource management operations *Seize* and *Release* in the simulation language GPSS for allocating and de-allocating (releasing) resources. Thus, GPSS has established a standard modeling pattern for resource-constrained activities, which has become popular under the name of *Seize-Delay-Release* indicating that for simulating a resource-constrained activity, its resources are first allocated, and then, after some delay (representing the duration of the simulated activity), they are de-allocated (released).

4.1.1 Resource Roles and Process Owners

As an illustrative example, we consider a hospital consisting of medical departments where patients arrive for getting a medical examination performed by a doctor in a room of the department. A medical examination, as an activity, has four participants: a patient, a medical department, a doctor and a room, but only two of them play a resource role: doctors and rooms. This can be indicated in an OEM class diagram by using the stereotype «resource role» for categorizing the association ends that represent resource roles, as shown in Figure 10.

Notice that both the event type *patient arrivals* and the activity type *examinations* have a (mandatory functional) reference property *process owner*. This implies that both patient arrival events and examination activities happen at a specific medical department, which is their process owner in the sense that it owns the process types composed of them. A process owner is called "Participant" in BPMN (in the sense of a collaboration participant) and visually rendered in the form of a container rectangle called "Pool".

In Figure 10, the resource role of doctors is designated as the *performer* role. Also in BPMN, *Performer* is considered to be a special type of resource role. According to Section 10.2.2 in the BPMN 2.0 specification (BPMN 2011), a performer can be "a specific individual, a group, an organization role or position, or an organization".

One of the main reasons for considering certain objects as resources is the need to collect *utilization statistics* (either in an operational information system, like a workflow management system, or in a simulation model) by recording the use of resources over time (their *utilization*) per activity type. By designating resource roles in information models, these models provide the information needed in simulations and information systems for automatically collect utilization statistics.

4.1.2 Resource Pools and Resource Allocation

In the hospital example, a medical department, as the process owner, is the organizational unit that is responsible for reacting to certain events (here: patient arrivals) and managing the performance of certain processes and activities (here: medical examinations), including the allocation of resources to these processes and activities. For being able to allocate resources to activities, a process owner needs to manage *resource pools*, normally one for each resource role of each type of activity (if pools are not shared among resource roles). A resource pool is a collection of resource objects of a certain type. For instance, the three X-ray rooms of a diagnostic imaging department form a resource pool of that department.

Resource pools can be modeled in an OEM class diagram by means of special associations between object classes representing process owners (like *medical departments*) and resource classes (like *doctors* and *rooms*), where the association ends, corresponding to collection-valued properties representing resource pools, are stereotyped with «resource pool», as shown in Figure 10. At any point in time, the resource objects of a resource pool may be *out of order* (like a defective machine or a doctor who is not on schedule), *busy* or *available*.

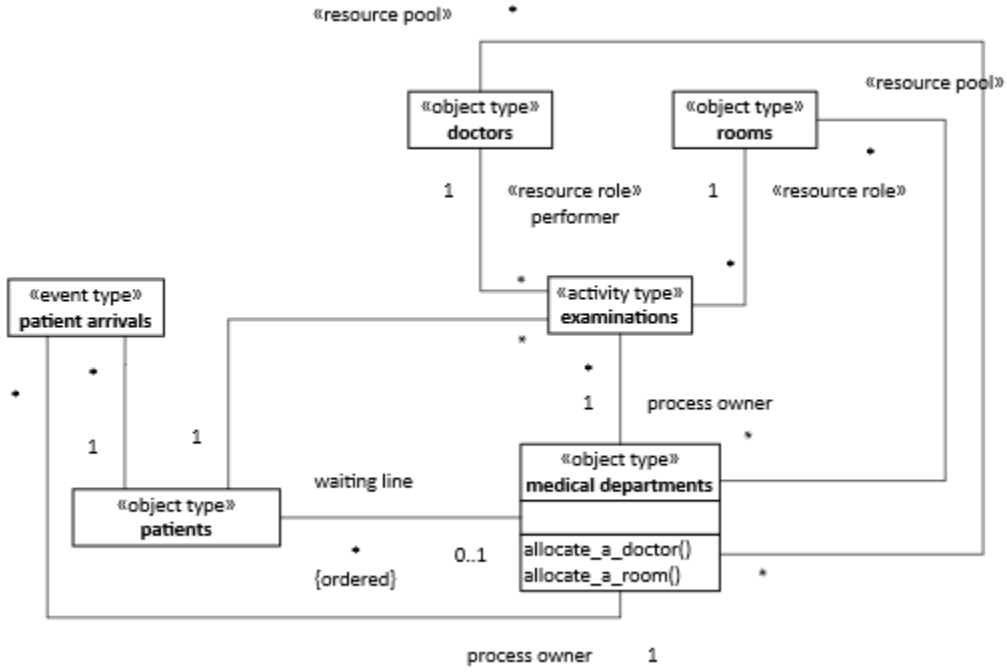


Figure 10: The activity type "examinations" with two resource roles and two resource pools.

A process owner has special procedures for allocating available resources from resource pools to activities. For instance, in the model of Figure 10, a medical department has the procedures "allocate a doctor" and "allocate a room" for allocating a doctor and a room to a medical examination. These resource allocation procedures may use various policies, especially for allocating human resources, such as first determining the suitability of potential resources (e.g., based on expertise, experience and previous performance), then ranking them and finally selecting from the most suitable ones (at random or based on their turn). See also (Arias et al 2018).

In the conceptual process model shown in Figure 11, a doctor and a room are always allocated and released (de-allocated) together.

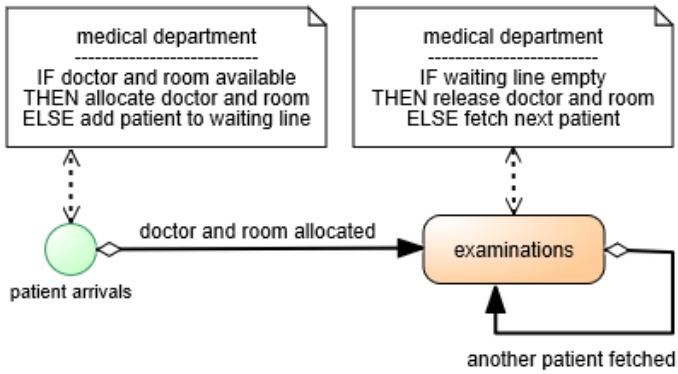


Figure 11: A conceptual process model based on the information model of Figure 10.

This process model describes two causal regularities in the form of the following two event rules, each stated with two bullet points: one for describing all the state changes and one for describing all the follow-up events brought about by applying the rule.

1. When a new patient arrives:
    - if a room and a doctor are available, then they are allocated to the examination of that patient; otherwise, if a room or a doctor is not available, the patient is added to the waiting line;
    - if a doctor and a room have been allocated, then start an examination of the patient.
2. When an examination is completed by a doctor in a particular room:
    - if the waiting line is empty, then the room and doctor are released; otherwise, if there are still patients in the line, the next patient is fetched to be examined by that doctor in that room;
    - if another patient has been fetched, then start the examination of that patient.

These conceptual event rules describe the real-world dynamics of a medical department according to business process management decisions. Changes of the waiting line and (de-)allocations of rooms and doctors are considered to be state changes (in the, not necessarily computerized, information system) of the department, as they are expressed in Data Object rectangles, which represent state changes of affected objects caused by an event in DPMN.

4.1.3 Queueing Planned Activities

Whenever an activity is to be performed but cannot start due to a required resource not being available, the *planned activity* is placed in a queue as a waiting job. Thus, in the case of a medical examination of a patient, as described in the model of Figure 10, the *waiting line* represents, in fact, a queue of planned examinations (involving patients), and not a queue of waiting patients.

As a consequence of these considerations, the *waiting line* of a medical department modeled in Figure 10 as an ordered collection of patients should be renamed to *planned walks*. In addition, a property *planned examinations*, which holds an ordered collection of patient-room pairs, should be added to the class *medical departments*. These model elements would reflect the hospital's business process practice to maintain a list of patients waiting for the allocation of a room to walk to and a list of planned examinations, each with a patient waiting for a doctor in an examination room.

4.1.4 Displaying the Process Owner and Activity Performers

The process owner and the involved performers can be displayed in a DPMN process diagram by using a rectangular *Pool* container for the process owner and Pool partitions called *Lanes* for the involved activity performers, as shown in Figure 12.

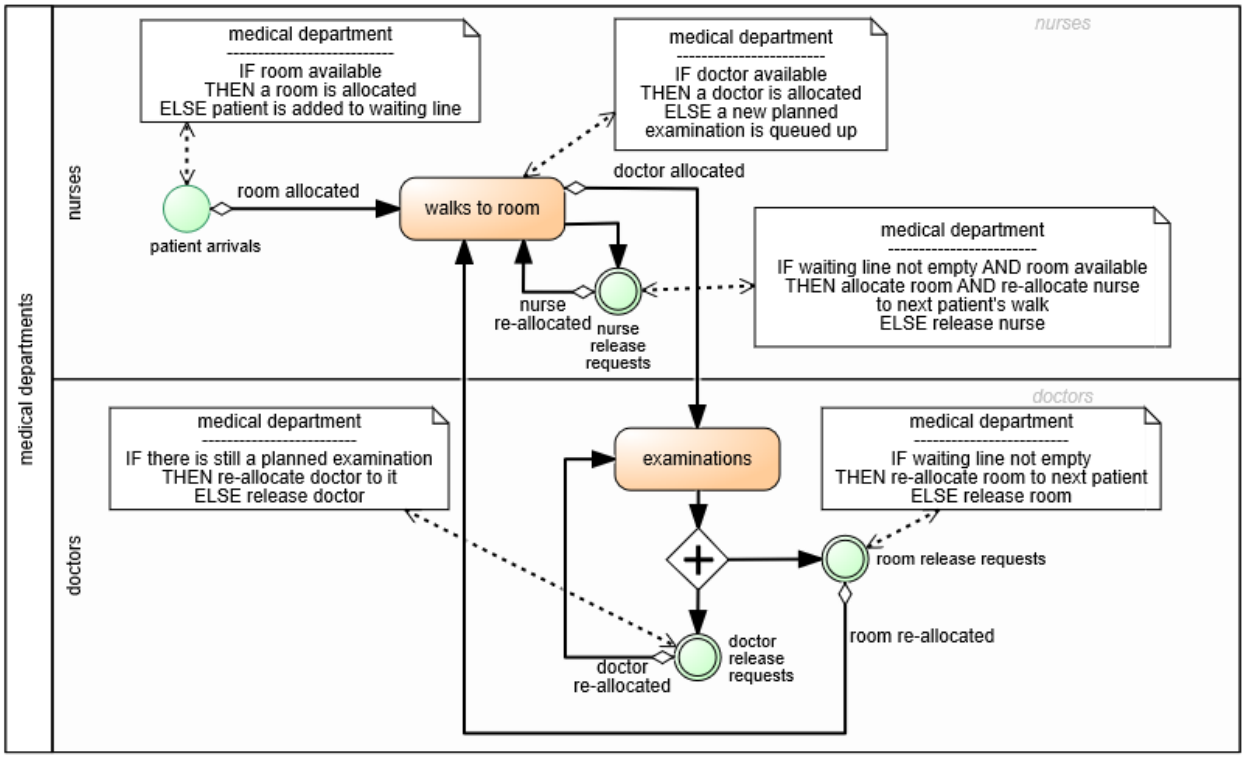


Figure 12: Displaying the process owner and activity performers in a conceptual process model.

4.2 Resource-Constrained Activities in Simulation Design Models

4.2.1 Extending OEM Class Diagrams by Adding a «resource type» Category

The conceptual information model of Figure 10 contains two object types, *rooms* and *doctors*, which are the range of resource role and resource pool properties (association ends stereotyped « resource role» and «resource pool»). Such object types can be categorized as «resource type» with the implied meaning that they inherit a resource *status* attribute (with possible values AVAILABLE, BUSY, OUT_OF_ORDER) and the resource management operations *isAvailable*, *allocate* and *release* from a pre-defined class *Resource*. The introduction of resource types allows simplifying models by dropping these modeling items from OEM-A class models, making them part of their implicit semantics.

4.2.2 Extending DPMN Process Diagrams by Adding Resource-Dependent Activity Start Arrows

The cluttering of process diagrams by displaying all the resource management logic required by resource-constrained activity types, as shown in Figure 12, can be avoided by introducing the new modeling element of *Resource-Dependent Activity Start (RDAS)* arrows, which combine event scheduling with the queuing of planned activities waiting for the availability of resources. For instance, in Figure 13, the intuitive meaning of the RDAS arrow between the *PatientArrival* event and the *WalkToRoom* activity is: when a

PatientArrival event has occurred, start a *WalkToRoom* activity (for walking the newly arrived patient to an examination room) as soon as the required resources (a room and a nurse) are available.

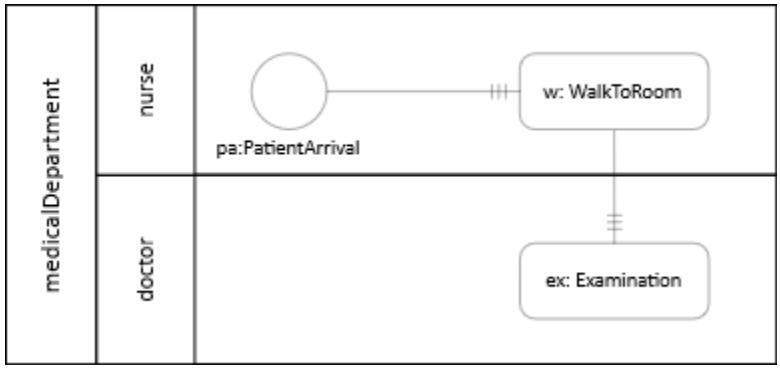


Figure 13: Using Resource-Dependent Activity Start Arrows in a process design model.

Notice that, as opposed to established "process-oriented" modeling tools, such as *AnyLogic*, the DPMN process model of Figure 13 does not need to specify any resource allocation/release steps, since they are implied by specifying the resource types and resource cardinality constraints of activity types in the underlying OEM-A class model.

Since DPMN's Resource-Dependent Activity Start Arrows, as shown in Figure 13, are not available in BPMN, new modeling tools will have to be developed for making DPMN Process Diagrams.

REFERENCES

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

Business Process Model and Notation (BPMN), Version 2.0, 2011. http://www.omg.org/spec/BPMN/2.0, accessed 13th May 2020.

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.

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

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

Wagner, G. 2019. "Information and Process Modeling for Simulation – Part II: Activities and Processing Networks". https://dpmn.info/reading/Activities.html, accessed 13th May 2020.

Wagner, G. 2018. "Information and Process Modeling for Simulation – Part I: Objects and Events". *Journal of Simulation Engineering* 1:1–25. https://articles.jsime.org/1/1.

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.. https://www.informs-sim.org/wsc17papers/includes/files/056.pdf.

AUTHOR BIOGRAPHY

GERD WAGNER is Professor of Internet Technology in the Dept. of Informatics, Brandenburg University of Technology, Germany, and Adjunct Associate Professor in the Dept. of Modeling, Simulation and Visualization Engineering, Old Dominion University, Norfolk, VA, USA. His research interests include modeling and simulation, foundational ontologies, knowledge representation and web engineering. In recent years he has developed the OEM&S paradigm and the process simulation modeling language DPMN (see https://dpmn.info). His email address is G.Wagner@b-tu.de.