Object Event Modeling for DES and IS Engineering

Gerd Wagner [1]

[1] *Brandenburg University of Technology, P. O. Box 101344, 03013 Cottbus, Germany*

Abstract

We propose Object Event Modeling as a new approach to Discrete Event Simulation (DES) engineering and Information Systems (IS) engineering using UML class diagrams for information modeling and BPMN-style process diagrams for process modeling. We consider the case where for a given business system, both a performance analysis is to be conducted with the help of a DES project and an IS is to be constructed with the help of an IS engineering project. Both the DES design model and the IS design model are representations of the same system. We argue that both DES modeling and IS modeling have a substantial overlap and, hence, can learn from each other. While research in DES has focused on events and processes, neglecting objects, IS engineering research has focused on objects, neglecting events.

Keywords

Information Systems Engineering, Discrete Event Simulation, Object Event Modeling

1. Introduction

The world consists of objects and events. “Smiles, walks, dances, weddings, explosions, hiccups, hand-waves, arrivals and departures, births and deaths, thunder and lightning: the variety of the world seems to lie not only in the assortment of its ordinary citizens—animals and physical objects, and perhaps minds, sets, abstract particulars—but also in the sort of things that happen to or are performed by them” [1].

Whereas research in *Discrete Event Simulation (DES)*, and in activity/flowchart-based *Business Process (BP) Modeling (BPM)*, has focused on events and processes, neglecting objects, research in *Conceptual Modeling (CM)* for *Information Systems (IS)* engineering and in data/object/entity-based BPM has focused on objects, neglecting events. Thus, both fields can learn from each other: DES, as well as flowchart-based BPM, could benefit from adopting object modeling concepts and techniques developed in IS engineering, and IS engineering, as well as data/object/entity-based BPM, could benefit from adopting event-based and process-based simulation concepts and techniques for extending its research scope and broadening its worldview.

The *Object Event Modeling* and *Simulation (OEM&S)* approach proposed by Wagner [14] includes a BPMN-style process modeling language, called the *Discrete Event Process Modeling Notation (DPMN)*. The visual syntax of DPMN is based on BPMN, while its (formal) semantics is based on Event Graphs [13], and not on Petri Nets.

In this paper, we show that a modeling approach based on objects and events, where objects participate in events and events cause state changes of participating objects and follow-up events, allows to model both the state structure and the dynamics of a discrete dynamic system such that the resulting model can be executed as a simulation of the system under investigation. In the proposed OEM&S approach, the state structure of a (simulation) system is described in the form of a UML class model defining *object types* and *event types*, and its dynamics is described in the form of a DPMN process model defining a set of *event rules*, which capture *causal regularities* and correspond to transitions of an *Abstract State Machine* [15].

ER'2022 Forum and PhD Symposium, October 17-20, 2022, Online
EMAIL: g.wagner@b-tu.de
ORCID: 0000-0003-3049-8885

Unlike in IS engineering, there are no widely accepted modeling languages for making conceptual models and design models in DES engineering [2]. Typically, in DES, neither conceptual nor design models are made, or they are made in a sketchy manner only, not using any well-defined modeling language, but rather simulation developers jump from their mental model, or sketch, to an implementation in some target technology platform, calling the resulting code “the simulation model”.

OEM&S is the first DES engineering approach based on conceptual modeling and design modeling using well-defined modeling languages (UML/OEM Class Diagrams and BPMN/DPMN Process Diagrams) as a basis for developing an executable simulation model. As opposed to a model as code, these models are more comprehensible and easier to communicate, share, reuse, maintain and evolve, while they can still be used for obtaining implementations for specific simulation platforms with the help of model transformations and code generation, as proposed in model-driven engineering.

Since both discrete event simulations and information systems are complex representations of real world systems, sharing the need of conceptual modeling as a prerequisite of design, it is natural to investigate the relationship between them. Concerning the various options for using IS and DES modeling, possibly in combination, we can distinguish the following cases:

1. Make an IS model for an organization that does not yet have an IS for automating its business processes.
2. Make a DES model for an organization that does not yet have an IS and wants to analyze its business processes.
3. Make a DES model of an organization’s business processes and then make an IS model by re-using as many DES modeling artifacts as possible.
4. First make an IS model for automating an organization’s business processes and then make a DES model of the IS-supported business processes for testing/evaluating the IS.

Consider the case where both a DES of an organization’s business processes and an IS supporting these processes are to be developed. Then we may ask questions like:

1. Can we use (a) the same conceptual modeling language and (b) the same conceptual model both for the DES sub-project and for the IS sub-project?
2. Can we use (a) the same design modeling language and (b) the same design model both for the DES sub-project and for the IS sub-project?
3. How can the two sub-projects benefit from each other?

OEM&S allows answering questions 1a and 2a with yes, and 1b as well as 2b with no. However, answering question 3 requires more research. It is startling that the relationship between these two related research areas, which have been developed largely independently of each other and have achieved many deep results during their histories of more than 60 years in the case of DES and more than 45 years in the case of CM for IS engineering, has not yet been investigated much.

2. Events in IS Engineering

As pointed out in [3], there have been a few proposals in the field of IS and Software Engineering in the 1980s and 1990s to model events as entities, but they have not been adopted in any standard modeling language (neither in UML, nor in BPMN). In BP modeling languages, which are typically based on the concept of activities, following the tradition of “flowcharts”, events (and activities) are not modeled as entities. This is not surprising since these languages are not concerned with modeling entities and only contain rudimentary concepts of entity types. But even in UML, with its Class Diagram language for modeling entity types, events are not modeled as entities. An exception is the conceptual modeling language of the *Object-Process Method (OPM)* [4], where process types, which can be considered as special types of events, are modeled as entity types (with attributes and operations).

The authors of [3] argue that modeling events as entities provides substantial benefits for CM. Based on, and strengthened by, the ontological analysis of the concept of events in [5, 6], we share their insight and extend it to the discipline of DES.

In [3], and in the field of DES, events are considered to be instantaneous. A general concept of events subsumes, however, both instantaneous events and events with durations (like walks or weddings), and the latter ones subsume activities as special types of events. Notice that this distinction is orthogonal to the distinction between atomic and composite events, as pointed out in [6]. An instantaneous event may be composed of several simultaneous component events, and an event with duration need not be composite.

Despite the fact that UML does not provide built-in support for modeling event types as classes in information models, it is clear that in many real-world IS, certain types of events are modeled and implemented as special classes having an attribute for the occurrence time of events. Examples of such events are *paper submissions* in the case of a conference organization IS or *car returns* in the case of a car rental IS.

In an IS, the current populations of object types represent the current state of affairs in the domain of the IS, as opposed to the populations of event types, which represent history information. Compare this with a running DES where the current state of the simulated system is also described by the populations of object types, while events are either currently occurring (being processed by the simulator) or scheduled to occur in the future when they are in the *Future Event List*.

In the field of information modeling for the Web, there have been various proposals how to represent event information on websites with the help of special vocabularies or information schemas like the *Event Ontology* [7] or the generic *Event* class defined by the search vocabulary *schema.org* [8] maintained by a consortium of web search vendors. Notice that in this field, the goal is not to define a generic event concept, but rather to define a feature-rich concept of social events, either historical or planned. Schema.org events have a start date/time, a duration and an end date/time.

The IS engineering approach of MERODE [18] takes *business events* into account as first-class citizens, along with business objects, for modeling the state structure and the behavior of a system. In MERODE, like in OEM&S, objects participate in events, which may create, update or destroy them, thus changing the state of an IS.

Remarkably, in the professional community of software engineers/architects, there has been a recent movement to complement OO modeling and programming with event-based approaches, such as *Event Sourcing* [19] and *Event Modeling* [20]. This shows that events are at the heart of IS engineering, and a more complete treatment of events, as provided by OEM&S, is a current topic.

The Enterprise Application Architecture pattern of *Event Sourcing* [19] ensures that all changes to the state of an IS (or Enterprise Application system) are stored as a sequence of events that can be queried and used for reconstructing past states. This is similar to the event log in DES. An Event Sourcing architecture supports building systems that need an audit trail.

Event Modeling [20] considers the business events created by user actions typically performed by pushing user interface buttons after providing data within form fields as fundamental.

3. Objects in DES Engineering

The term *Discrete Event Simulation (DES)* has been established as an umbrella term subsuming various kinds of computer simulation approaches, all based on the general idea of modeling a discrete dynamic system by modeling its state as a set of state variables, and modeling its dynamics by modeling the events that are responsible for its state changes and how they do this. There is, however, no generally accepted definition of DES. Simulation textbooks and tutorials avoid defining the term “DES” in a precise way.

Pegden [9] describes three fundamental DES paradigms, which have shaped the 60-year history of DES: Markowitz, Hausner, and Karr [10] pioneered *Event-Based Simulation* with *SIMSCRIPT* (1962), Gordon [11] pioneered *Processing Network Simulation* with *GPSS* (1961), and Dahl and Nygaard [12] pioneered *Object-Orientation (OO)* and the computational concept of *co-routines* for asynchronous programming with *Simula* (1967).

In the *Event-Based Simulation (ES)*, or *Event Scheduling*, paradigm, the system under investigation is viewed as a series of instantaneous events that change the state of the system over time. The modeler defines the system’s state structure with the help of *state variables* and models the system’s state changes by defining the types of events that occur in the system and the state changes of affected state variables caused by their occurrences.

In the *Processing Network Simulation (PNS)* paradigm, the system under investigation is described as a processing network where “entities flow through the system” or, more precisely, *processing objects* are routed through the network and are subject to a series of processing steps performed at processing nodes through *processing activities*, typically requiring *resources* and inducing *queues* of processing objects waiting for the availability of resources. This approach allows high-level modeling with semi-visual languages and is therefore the most widely used DES approach today, in particular in manufacturing industries and service industries. Simulation software packages based on this paradigm may or may not support OO modeling and programming.

According to Pegden [9], 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, 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.

According to Pegden [9], ES is the most fundamental paradigm since other DES paradigms also use events, at least implicitly. However, ES does not include the modeling concepts of *objects* and *classes*, which have been introduced some years later with the simulation language Simula and have become popular in the field of information technology only in the 1980s and 90s with the programming languages Smalltalk, C++ and Java and the OO modeling language UML.

After OO had been established as the predominant paradigm in software engineering in the 1990s, it was also adopted by many simulation tools, which have used it as the basis of their PNS approach. Consequently, modern DES tools allow combining PNS models with OO modeling and some form of event scheduling.

In BPM, the modeling concept of *objects* (with attribute-value slots and links to other objects) is neither well supported in Petri-Net-based approaches nor in BPMN. It is common to make the simplifying assumption that a task requires just one resource object and a resource object cannot be used in more than one task at the same time. Due to the resulting simplistic concept of business processes, BPMN-based business process simulation (e.g., in Signavio) is very limited and cannot compare to business process simulation in DES.

4. From Event Graphs via Object Event Graphs to Activity Networks

The visual modeling language of *Event Graphs* proposed by Schruben [13] in 1983 allows specifying ES models, and its formal semantics provides a formal semantics of ES. While Event Graphs have been adopted as a modeling language to some degree in the field of DES, mainly in Operations Research, they have been ignored in Computer Science and Information Systems, including the field of BPM, despite the fact that they are a process modeling language with a formal semantics supporting parallelism, like Petri Nets. 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.

DPMN is incrementally constructed from Event Graphs by adding increasingly high-level modeling concepts in two steps: in the first step, the concept of objects is added, resulting in *Object Event Graphs*, and in the second step, the concept of (resource-constrained) activities is added, resulting in *Activity Networks*. Both Object Event Graphs and Activity Networks are special forms of DPMN process models. Their execution semantics is based on the concept of an *Object Event (OE) model* of a system under investigation, which is a triple $\langle OT, ET, R \rangle$ where

1. *OT* is a set of object types defining the state structure of the system;
2. *ET* is a set of event types;
3. *R* is a set of event rules (expressed in terms of *OT* and *ET*) defining the dynamics of the system, such that R contains a rule for each event type in *ET*.

An OE model, together with an initial state (consisting of initial objects and initial events), defines an *Object Event Simulation (OES)* system, which is a transition system where

1. the state of a simulation consists of two parts: (a) a system state given by the union of the sets of all attribute-value slots of all objects, and (b) a set of time-stamped events called the *Future Events List (FEL)*;

2. transitions are provided by event occurrences triggering event rules that change the simulation state by changing the system state (by changing the states of affected objects) and the FEL (by adding follow-up events).

We have shown in [15] that the transition system defined by an OES system is an *Abstract State Machine (ASM)* in the sense of Gurevich [16].

Instead of defining a new modeling language for OEM&S, it is preferable to re-use languages that are already well-established, such as UML and BPMN, and modify them if necessary. Both the object types *OT* and the event types *ET* of an OE model can be defined as classes of an information model in the form of an OE class diagram. The event rules *R* can be defined as an OE Graph that is decomposable into a set of OE Graphs, one for each event circle representing a triggering event variable and defining an event rule, such that the rules' conditions, state change statements and follow-up event scheduling expressions may refer to the elements defined in the underlying OE class model.

4.1. 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).

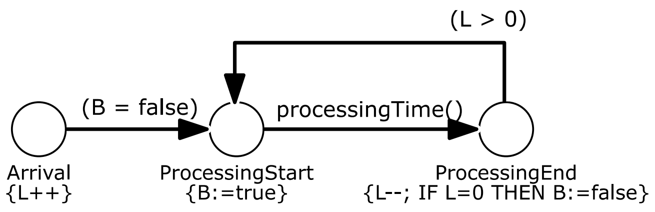


Figure 1: An Event Graph defining an ES model with two state variables and three event types.

Event Graphs provide a visual modeling language with a precise semantics that captures the fundamental ES paradigm. However, Event Graphs are a rather low-level DES modeling language: they lack a visual notation for (conditional and parallel) branching, do not support OO state structure modeling (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 («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.

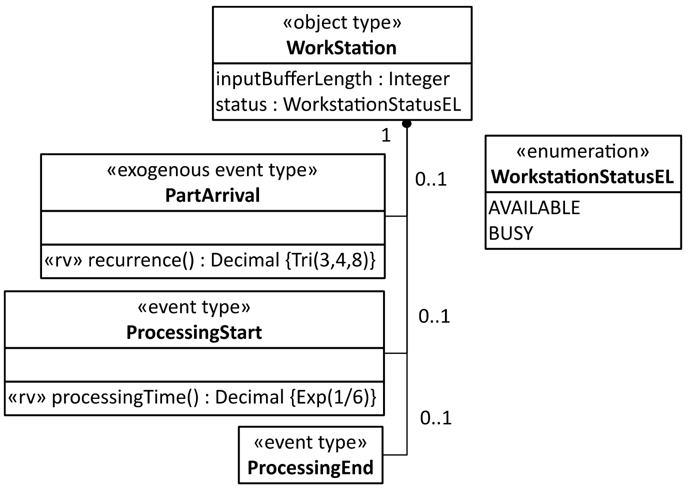


Figure 2: An OE class model defining an object type and three event types.

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].

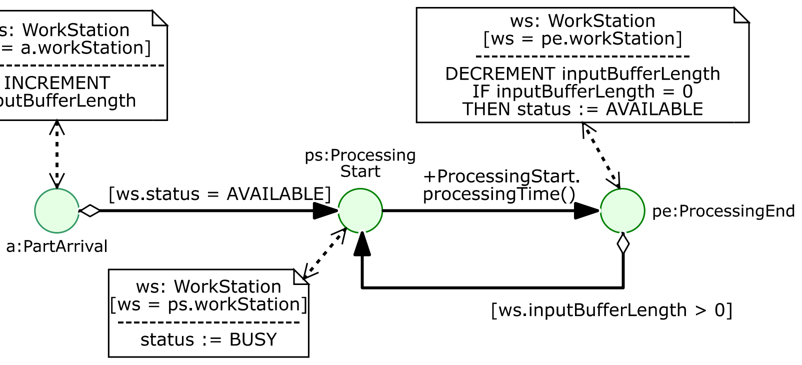


Figure 3: An Object Event Graph based on the OE class model of Figure 2.

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 modeling OE Graphs, corresponds to an extension of Event Graphs by adding the concept of *Objects*. The second layer (DPMN-A), for modeling 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 modeling Processing Networks, adds the concepts of *Processing Objects*, *Processing Activities* and *Processing Flows*.

4.2. From Object Event Graphs to Activity Networks

In [17], we have shown how to extend OE Graphs by adding support for resource-constrained activities, resulting in OEM/DPMN-A, comprised of three new information modeling elements (*Activity Type*, *Resource Role*, *Resource Pool*) and two new process modeling elements (*Activity* and *Resource-Dependent Activity Scheduling Arrow*). DPMN-A diagrams allow modeling *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 OE class model of Figure 4 below, the pair of *ProcessingStart* and *ProcessingEnd* event types of the 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.

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.

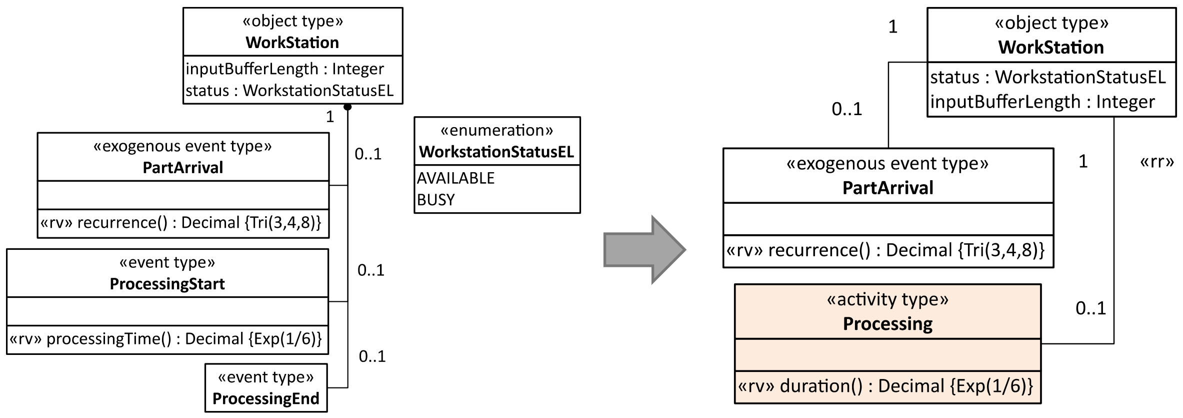

«object type»
WorkStation
inputBufferLength : Integer
status : WorkstationStatusEL
1
0..1
«exogenous event type»
PartArrival
«rv» recurrence() : Decimal {Tri(3,4,8)}
«enumeration»
WorkstationStatusEL
AVAILABLE
BUSY
«event type»
ProcessingStart
«rv» processingTime() : Decimal {Exp(1/6)}
0..1
«event type»
ProcessingEnd
0..1
1
«object type»
WorkStation
status : WorkstationStatusEL
inputBufferLength : Integer
0..1
«exogenous event type»
PartArrival
«rv» recurrence() : Decimal {Tri(3,4,8)}
1
«rr»
«activity type»
Processing
«rv» duration() : Decimal {Exp(1/6)}
0..1

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 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).

5. Case Study: Making an Object Event Model of a Public Library

We consider the case of a public library as an example of a business system, for which we make both an OES model and an IS model for investigating the research questions posed in the Introduction.

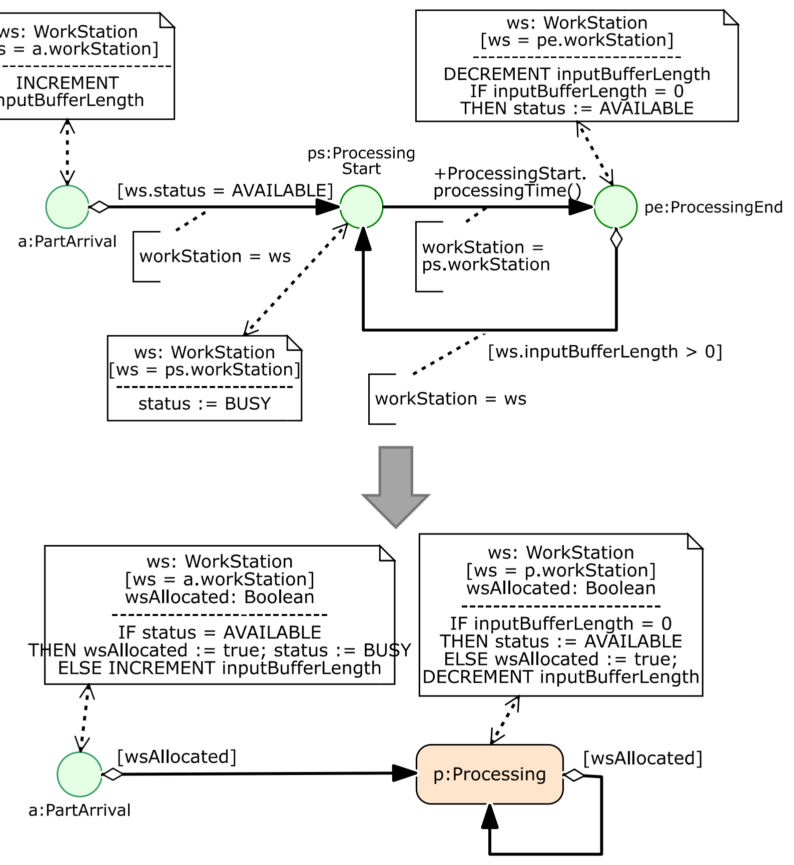


Figure 5: Rewriting an OE Graph to a corresponding Activity Network model based on the target OE class model of Figure 4.

5.1. Making an IS Model

As a result of requirements and domain analysis, we have obtained the conceptual information model shown in Figure 6, describing 9 entity types with attributes and associations.

The 9 entity types described by this model include 3 potential event types: book pickup notifications, book lendings and book returns. However, from an IS modeling point of view, these entity types may also be viewed as types of objects that represent documents recording events. This interpretation (of events as document objects) is, in fact, supported by the presence of date/time attributes in these entity types and by the *many* multiplicity in their participation associations with object types.

Observation 1: Since, ontologically, ***objects participate in events***, the associations between object types and event types are *participation associations*.

Compare the one-to-many association between *library users* and *book returns* in Figure 6 with the one-to-one association between *WorkStation* objects and *PartArrival* events in Figure 4. When an event class is populated with current events only (according to the standard snapshot semantics of UML class models), a participation association between an object class and that event class is functional (x-to-one), while it is non-functional (x-to-many) in the case of an event document class populated with an event history (in the form of objects representing event documents).

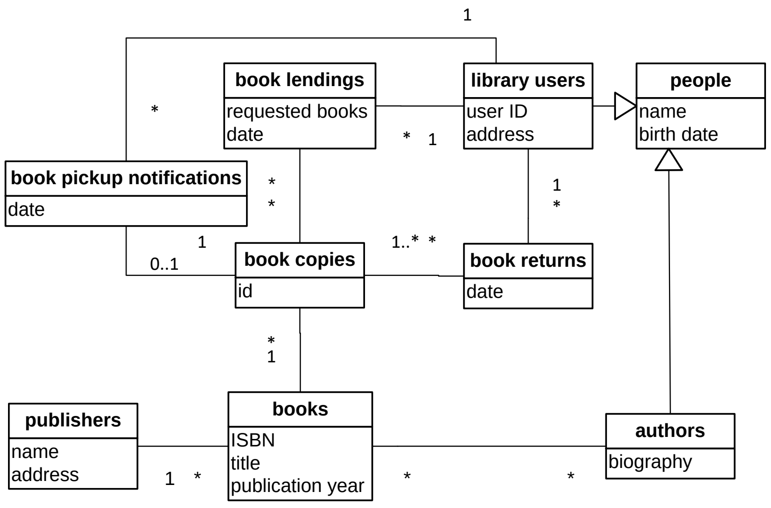


Figure 6: A conceptual information model for an IS for the public library.

Observation 2: An information model for an IS typically represents event types in the form of event document object classes for recording event histories, while an information model for an OES represents event types as classes with snapshot populations. In the case of an IS model, participation associations are non-functional, while they are functional in the case of an OES model.

An information design model for an IS for the public library based on the conceptual information model of Figure 6 is shown in Figure 7.

Notice that while historic lending events are stored in the population of *BookLending*, current lendings, which still need to be completed, are recorded in the multivalued *Book* attribute *lendings*. This allows the IS to periodically check if users have to be reminded to return lended books. The multivalued *Book* attribute *reservations* allows to check if a returned book copy has to get the status RESERVED and the corresponding user has to be notified to pick up the reserved book.

5.2. Making an OES Model

The conceptual information model shown in Figure 8 describes the relevant entity types of an OES model, where the date/time attributes of the event types described in the IS model of Figure 6 have been dropped and the *many* multiplicities of the participation associations described in the IS model of Figure 6 have been replaced with 0..1 multiplicities.

Notice that in addition to the three event types described by the IS model of Figure 6, the OES model in Figure 8 includes two further event types, which are essential for capturing the dynamics of a library as a business system: *user arrivals* and *user departures*. These two event types, which are important for capturing user behavior in the simulation model, would normally only be included in an IS model, if the IS is connected to member card scanners for automatically recording the entry and exit of users.

While user arrivals and book pickup notifications are instantaneous events, book lendings and book returns may be modeled either as instantaneous events or as activities. This consideration explains that we have a choice to make an OE Graph or an Activity Network as a process model for the public library. We start with an OE Graph in Section 5.2.1 followed by an Activity Network in Section 5.2.2.

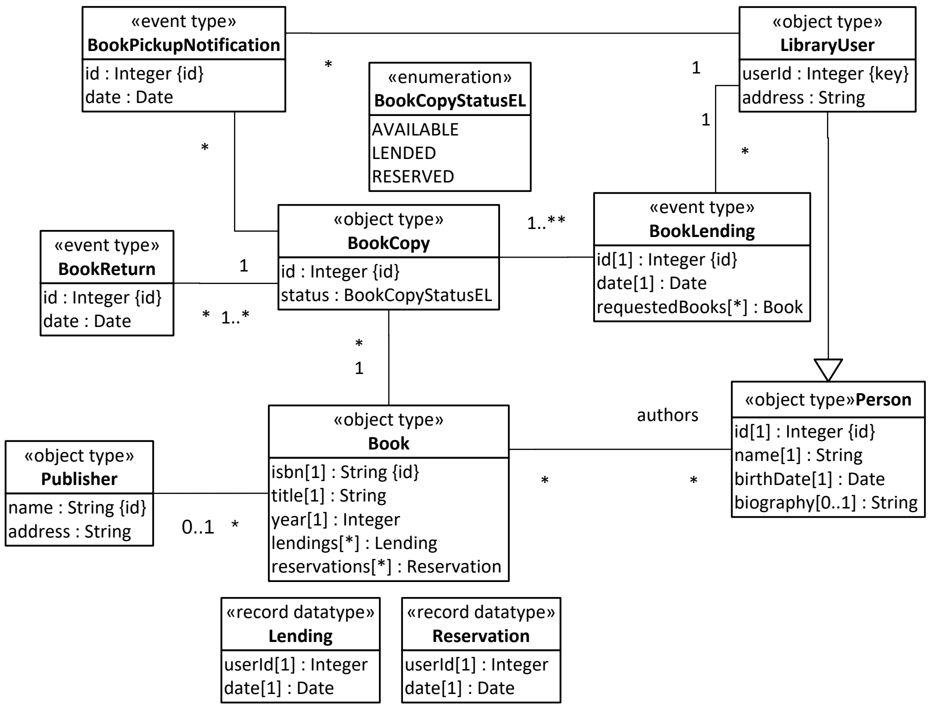


Figure 7: An information design model for a public library IS.

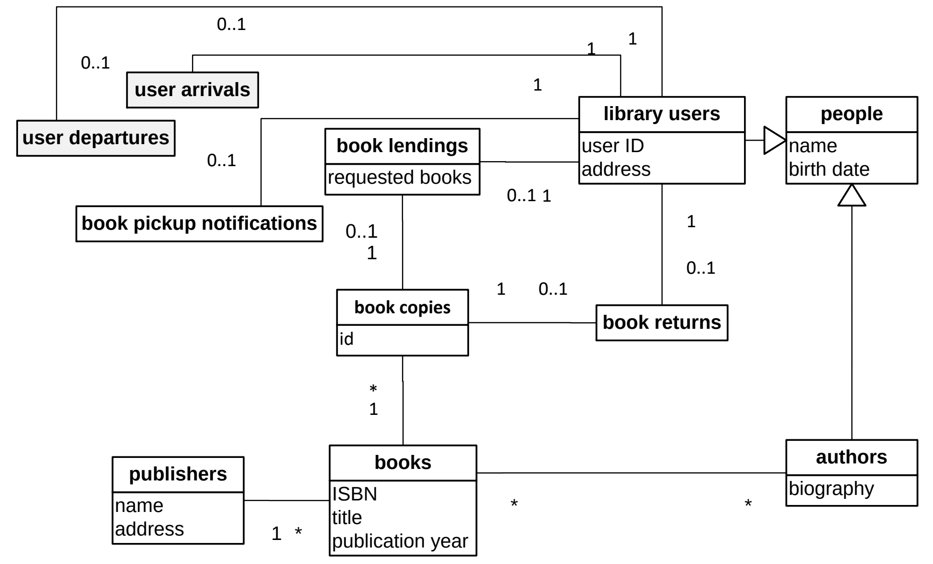


Figure 8: A conceptual information model for an OES model of the public library.

5.2.1. Modeling the Public Library Process as an OE Graph

Based on the conceptual information model for OES shown in Figure 8, we can now make a conceptual process model for an OE simulation of the public library in the form of an OE Graph as shown in Figure 9.

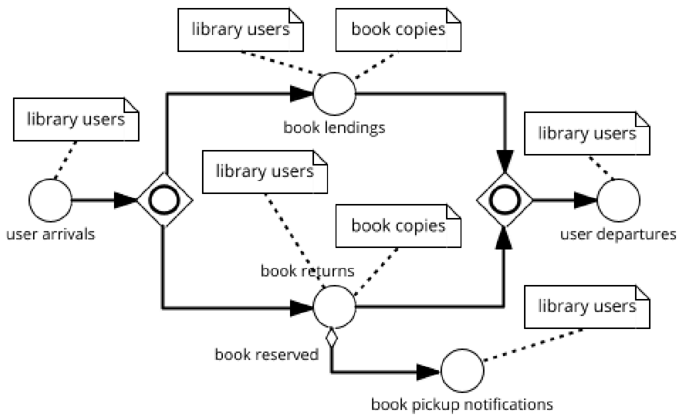


Figure 9: An OE Graph as a conceptual process model for an OE simulation of the public library.

This model describes the possible flows of events, assuming that book lendings and book returns are instantaneous events. When users arrive, two events may happen in any order, as indicated by the *Inclusive OR Gateway*: they may return any number of books in a *book return* event, or they may lend any number of books in a *book lending* event. When a book is returned for which a reservation has been made, then a *book pickup notification* event happens.

5.2.2. Modeling the Public Library Process as an Activity Network

Modeling book lendings and book returns as *instantaneous events* abstracts away from the duration these events have in reality and does not allow modeling the library's service desk involved in these events as a *resource* that can be busy and thus create waiting lines. In a more realistic model, these events should be modeled as *activities* depending on an available service desk as a resource.

The two event types book lendings and book returns can also be modeled as activity types, as in Figure 10.

Compared to the model of Figure 9, turning *book lendings* and *book returns* into activities implies that they have some *duration* as the time between their (implicit) start and end events. Typically, in an OES model, the duration of activities of a certain type is defined in the form of a random variable (with an underlying probability distribution) in an OE class model.

This model describes the possible flows of events and activities. When users arrive, they can return any number of books with a *book return* activity, or they can lend any number of books with a *book lending* activity.

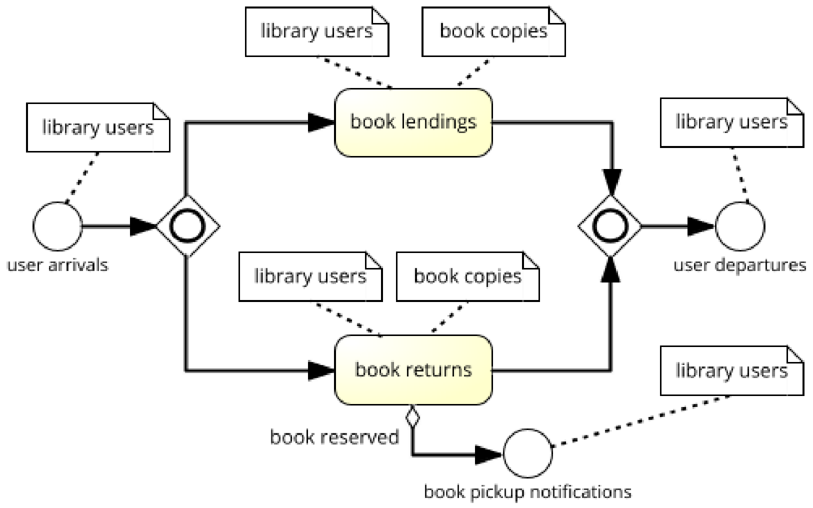


Figure 10: A conceptual process model for the public library in the form of a DPMN Activity Network.

While we can use the same information design modeling language both for the IS project and for the OES project, we cannot use the same information design model for both projects. Rather, based on Observation 2, we can derive the OES class design model from the IS class design model by

1. Dropping the event types' ID and date/time attributes, since these attributes are implicit in OE class models.
2. Changing the event multiplicities of participation associations from * to 0..1.
3. Adding those further event types that are essential for capturing the dynamics of the system under investigation.
4. Designating the exogenous event types, for which a recurrence function has to be specified (typically representing a random variable).
5. Defining a random variable duration function for all activity types.

The result is the OE class design model shown in Figure 11.

Notice that, like in the IS model of Figure 7, the *Book* attribute *reservations* allows checking if a returned book copy has to get the status RESERVED and a *BookPickupNotification* event has to be created for notifying the corresponding user. The attribute *lendings* has been moved from *Book* to *LibraryUser* where it allows the OES to periodically check if users have to go to the library for returning lended books.

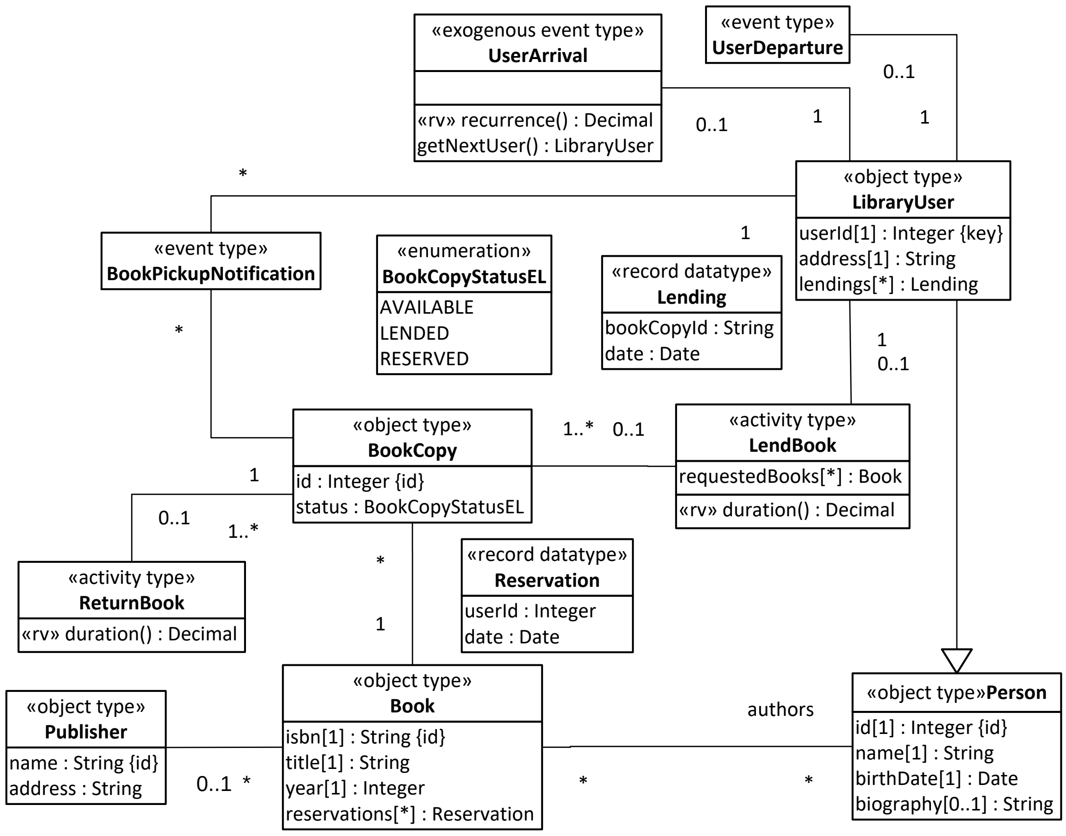
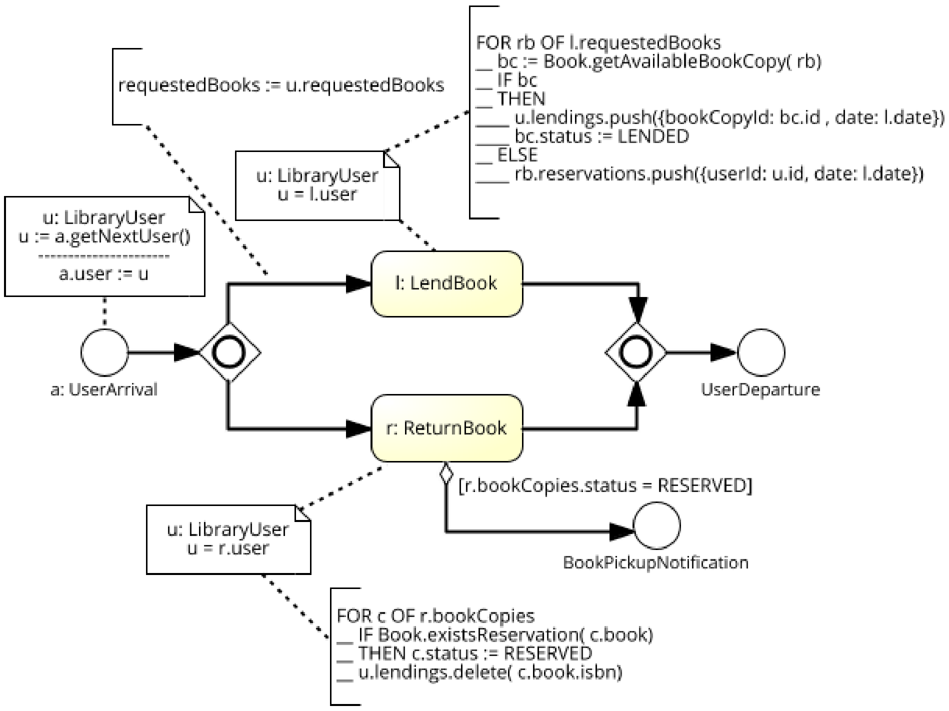


Figure 11: An OE class design model as the basis of an Activity Network design model.

While in the library IS, the periodic checks if book copies need to be returned are performed by the library, which sends out reminder messages to the users concerned, the library OES takes care of this by modeling users that periodically check their lending records for making sure that they don't miss a due date. In the *UserArrival* event class, a *getNextUser* method has been added for selecting a user, either on the basis of a current due date or at random. This method is also in charge to create a list of requested books (at random) to be passed to the follow-up *LendBook* activity.

Finally, the Activity Network design model shown in Figure 12 is derived from the conceptual process model of Figure 10 on the basis of the OE class design model of Figure 11 by expressing the event rules for *UserArrival* events and for *LendBook* and *ReturnBook* activity end events, such that these rules are triggered on completion of these activities. For instance, the *LendBook* activity end event rule is specified by (1) the *l:LendBook* activity rectangle declaring the activity variable *l* as representing the current *LendBook* activity; (2) the *u:LibraryUser* object rectangle introducing the activity variable *u* as bound to the user object referenced by the expression *l.user*, which denotes the user involved in the current *LendBook* activity; (3) the state change code of the annotation attached to the object rectangle; and (4) a follow-up event of type *UserDeparture*.



```
FOR rb OF l.requestedBooks
__ bc := Book.getAvailableBookCopy( rb)
__ IF bc
__ THEN
____ u.lendings.push({bookCopyId: bc.id , date: l.date})
____ bc.status := LENDED
__ ELSE
____ rb.reservations.push({userId: u.id, date: l.date})
```

```
FOR c OF r.bookCopies
__ IF Book.existsReservation( c.book)
__ THEN c.status := RESERVED
__ u.lendings.delete( c.book.isbn)
```

Figure 12: A DPMN process design model in the form of an Activity Network.

6. Using OEM&S and DPMN for IS Engineering

While OEM&S and DPMN have been conceived for DES engineering, due to the large overlap between DES engineering and IS engineering, they can also be used for IS engineering. While the activities in a DPMN process model for a DES engineering project are supposed to be executed by a simulator (based on their *duration* random variable), the activities in a DPMN process model for an IS engineering project are to be executed by (typically human) performers, which requires that they can interact with the IS for getting information from the IS and for providing information to the IS via a suitable user interface (UI) that can be dynamically generated for each activity type on the basis of the underlying OE class model and DPMN process model. When the user completes the activity by a performing a suitable UI action (such pushing a button), the IS can execute the activity's state change statements and initiate any follow-up activity user interactions or perform any follow-up action events specified for it in the DPMN process model. This is similar to what a "process-aware" IS based on the workflow management paradigm is doing, but its operational semantics is more general, since it is based on Object Event Graphs [15].

The semantics of OE class models and of queries/conditions in OEM&S and DPMN are based on predicate logic. Each object type/class and each event type/class defines a corresponding unary predicate, while their attributes define binary predicates. A condition expressed in terms of these predicates holds in a system state when the state's populations of all object classes form a model that satisfies the condition.

The formal semantics of DPMN process models (in the form of OE Graphs and Activity Networks) is provided by mapping their implicit event rules to transitions of an Abstract State Machine, see [15]. As opposed to Petri-Net-based BPM approaches, such as BPMN, the OEM&S/DPMN approach is ontologically well-founded on the ontological categories of objects, events and causal regularities.

7. Related Work

We briefly discuss the most relevant related works: UML State Machines, MERODE, OPM, and data/object/entity-based BPM. Notice that, as opposed to OEM&S/DPMN, neither of them supports event-based simulation modeling.

UML State Machine diagrams describe finite state automata in terms of named object states and state transitions, which are triggered by message (or operation call) events. They are based on Harel's *Statecharts* formalism [21] and typically used for defining the behavior of the instances of an object class. In State Machines, a State models a situation during which some invariant condition holds. In most cases this condition is not explicitly defined, but is implied, usually through the name of the State. A Transition represents a specific form of event rule: it is bound to a source State and leads to a target State, if it is triggered by a specific message event and its 'guard' condition holds.

Compared to OEM&S/DPMN, UML State Machines are based on the abstract computational concepts of States and Transitions, and not on the fundamental ontological categories of Objects, Events, and Causal Regularities. They do not include a general concept of events as first class citizens and they require to identify a small number of relevant object states that can be named for forming the States of the State Machine.

Two well-established approaches resulting from IS engineering research, OPM [4] and MERODE [18], take events or processes into account as first-class citizens, along with objects, for modeling the structure and behavior of a system.

The IS engineering approach of MERODE [18] assumes that each type of object participating in an event of type *evt* defines an operation with the (same) name *evt*, such that at execution time, when an event of type *evt* occurs, the *evt* operations of all participating objects are invoked. This approach to behavior/process modeling, called 'multiparty interaction', has been adopted from process algebras like CSP [22] where the term 'events' stands for inter-process message events. In MERODE, it takes the form of *Object-Event Tables*, which express, for each business event type, which objects are affected in which way (Create/Update/Destroy) by the occurrence of an event of that type, and is combined with UML State Machines, where events are the triggers for transitions from one named state to another.

However, MERODE's Object-Event Tables allow only to capture restricted forms of event rules that do not allow to specify follow-up events and require to split up a rule's cross-object state change statements into a set of object-specific state change statements, while the use of UML State Machines requires the identification of a small number of named states, which may be difficult and artificial in many cases.

The IS engineering approach of OPM [4] allows the integrated modeling of a system's structure and behavior. The basic building blocks of OPM are objects (with named states) and processes.

However, OPM only has a limited concept of events (as provided by an object entering/exiting a named state or a process starting/ending, not modeling event types as first class citizens), does not distinguish between activities and processes, and requires, like State Machines, to identify a small number of relevant object states that can be named. Also, the Event-Condition-Action (ECA) rules of OPM represent a restricted form of OEM's event rules, which split up the Action part into state change statements and follow-up events for creating an explicit trace of event causation.

There are several proposals for data/object/entity-based BPM, often motivated by allowing more flexible types of BPs. E.g., the approach of [23] is based on *Business Entities with Lifecycles (BELs)*, previously also called 'artifacts', where lifecycle stages are similar to the named object states in State Machines and OPM. The operational semantics of this approach is based on ECA rules, like in OPM, but its process models are not visual, as in DPMN, but purely textual (consisting of ECA rule code).

8. Conclusions

In this paper, we have started to investigate the complementary model-based engineering of a DES and of an IS for a given business system. By allowing to capture general forms of event rules in OE Graphs or Activity Networks, DPMN process models can be used to model the dynamics of an IS by focusing on business events/activities and their causal regularities.

9. References

[1] Casati, R., and A. Varzi, Events, in: E.N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, 2015, URL: http://plato.stanford.edu/archives/win2015/entries/events/

[2] Robinson, S., G. Arbez, L.G. Birta, A. Tolk, and G. Wagner, Conceptual Modeling: Definition, Purpose and Benefits, in L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti (Eds.), Proc. of the 2015 Winter Simulation Conference, IEEE, 2015.

[3] Olivé, A., and R. Raventós, Modeling events as entities in object-oriented conceptual modeling languages, Data & Knowledge Engineering 58:3, 2006, 243–262.

[4] Dori, D. Object-Process Methodology: A Holistic Systems Paradigm. Berlin, Germany: Springer, 2002.

[5] Guizzardi, G. and G. Wagner. 2013. "Dispositions and Causal Laws as the Ontological Foundation of Transition Rules in Simulation Models". In Proceedings of the 2013 Winter Simulation Conference, in R. Pasupathy et al (Eds.) , 1335–1346, IEEE.

[6] Guizzardi, G., G. Wagner, R. Falbo, J.P. Almeida and R. Guizzardi. Towards Ontological Foundations for the Conceptual Modeling of Events. In: J.C. Trujillo, W. Ng and V. Storey (Eds.), *Proceedings of 32nd International Conference on Conceptual Modeling*, 2013.

[7] Raimond, Y. and S. Abdallah, The Event Ontology, 2007, accessed on July 25, 2022, URL: http://motools.sourceforge.net/event/event.html.

[8] Schema.org website, accessed on June 26, 2022, URL: https://schema.org/Event

[9] Pegden, C.D., Advanced Tutorial: Overview of Simulation World Views, in B. Johansson et al (Eds.), Proceedings of the 2010 Winter Simulation Conference, 643−651, IEEE, 2010.

[10] Markowitz, H., B. Hausner, and H. Karr, SIMSCRIPT: A simulation programming language, Englewood Cliffs, N. J.: Prentice Hall, 1962.

[11] Gordon, G., A general purpose systems simulation program, in Proceedings of the Eastern Joint Computer Conference, Washington, D.C., 1961.

[12] Dahl, O.-J. and K. Nygaard, Simula 67, *IFIP TC 2 Working Conference on Simulation Languages*, Oslo, 1967.

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

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

[15] Wagner, G., An Abstract State Machine Semantics for Discrete Event Simulation. In W. K. V. Chan et al (Eds.), *Proceedings of the 2017 Winter Simulation Conference*, 762–773, IEEE, 2017, URL: https://www.informs-sim.org/wsc17papers/includes/files/056.pdf.

[16] Gurevich, Y., A new thesis, American Mathematical Society Abstracts, page 317, August 1985.

[17] Wagner, G., Business Process Modeling and Simulation with DPMN: Resource-Constrained Activities, In K.-H. Bae et al (Eds.), *Proceedings of the 2020 Winter Simulation Conference*, 762–773, IEEE, 2020.

[18] Snoeck, M., Enterprise Information Systems Engineering – The MERODE Approach, Springer Cham, 2014.

[19] Fowler, M., Event Sourcing, web page, 12 December 2005, accessed on July 25, 2022, URL: https://martinfowler.com/eaaDev/EventSourcing.html.

[20] Dymitruk, A., "Event Modeling: What is it?", blog post, 2019, accessed on July 25, 2022, URL: https://eventmodeling.org/posts/what-is-event-modeling/.

[21] Harel, D., Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming* 8(3), 231–274, 1987.

[22] Hoare, C.A.R., Communicating sequential processes, Prentice-Hall International, Series in Computer Science, 1985.

[23] Hull, R., et al, Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles, in Proc. Int. Workshop on Web Services and Formal Methods (WS-FM), Springer-Verlag LNCS 6551, 2010.