OBJECT EVENT SIMULATION (OES)

A New DES Paradigm with a Formal Semantics

GERD WAGNER

BRANDENBURG UNIVERSITY OF TECHNOLOGY

GERMANY

This SIMULTECH 2020 presentation is available from **https://dpmn.info/reading/SIMULTECH2020**.

OVERVIEW

- Prolog: Conceptual Confusion in Discrete Event Simulation (DES)
- Part I: Discrete Dynamic Systems
- Part II: Information Modeling with UML Class Diagrams
- Part III: Process Modeling with Extended Event Graphs
- Part IV: Simulation with OESjs

PROLOG

CONCEPTUAL CONFUSION IN DES

ARE YOU ALSO CONFUSED ABOUT...

- What is Discrete Event Simulation (DES)?
- What is an Activity?
- What is a Process?
- What is Process-Oriented Simulation?
- What is Process-Interaction Simulation?
- What is an agent, as opposed to an object? What is Agent-Based Simulation?
- Where are the objects, and why is there no OO Modeling, in DES?
- Why is there no standard modeling language in DES, except Event Graphs (Schruben 1983)?
- Why are Event Graphs hardly used?

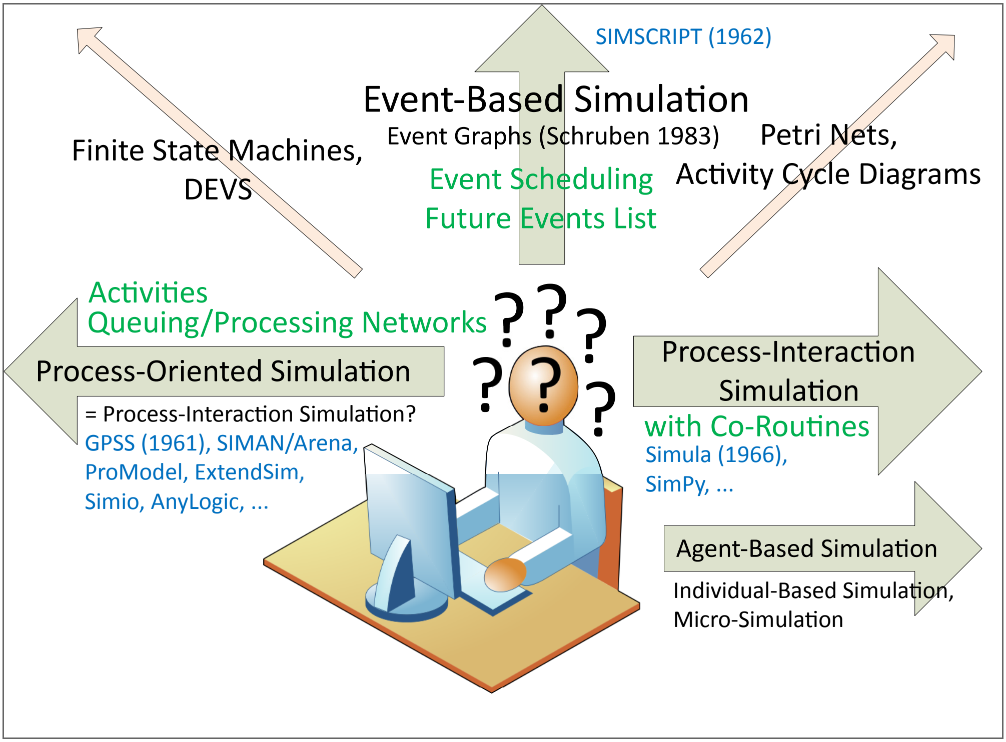
SOME OBSERVATIONS

- There is a lot of conceptual confusion in DES.
- DES textbooks avoid defining DES.
- Event-Based Simulation, as defined by Event Graphs, is the foundation of DES.
- All other DES languages/frameworks should extend Event-Based Simulation.
- Activities, as an important high-level modeling concept, should be defined on top of events.
- "Process-Oriented" Simulation is, in fact, about Queuing/***Processing Networks***.
- Processing Network models ("entities flowing through a system") are a special class of DES models.

MODELING LANGUAGE USAGE

Modeling Language	**BPM**	**DES**
Petri Nets (1939)	+	-
Event Graphs (1983)	--	+
UML Activity Diagrams (1997)	+	--
BPMN (2004)	++	-
UML Class Diagrams (1997)	-	-

CONCEPTUAL CONFUSION



Event-Based Simulation

SIMSCRIPT (1962)

Event Graphs (Schruben 1983)

Event Scheduling

Future Events List

Finite State Machines,
DEVS

Petri Nets,
Activity Cycle Diagrams

Activities
Queuing/Processing Networks

Process-Oriented Simulation

= Process-Interaction Simulation?

GPSS (1961), SIMAN/Arena,
ProModel, ExtendSim,
Simio, AnyLogic, ...

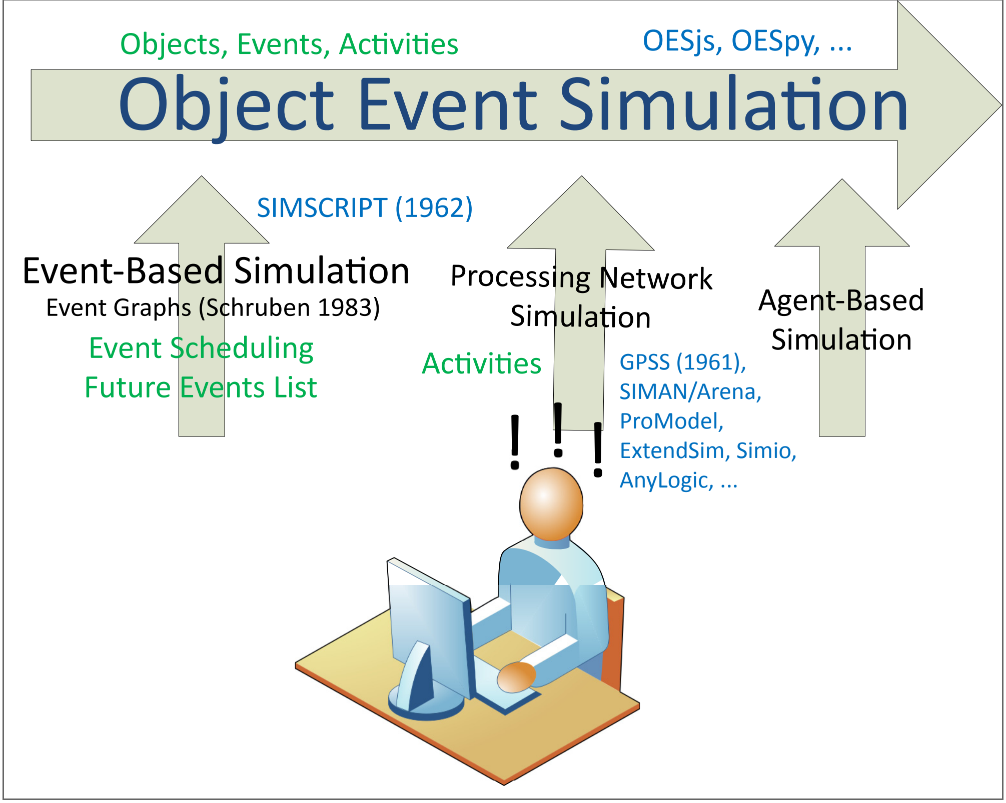
Process-Interaction
Simulation

with Co-Routines

Simula (1966),
SimPy, ...

Agent-Based Simulation

Individual-Based Simulation,
Micro-Simulation



Objects, Events, Activities

OESjs, OESpy, ...

Object Event Simulation

SIMSCRIPT (1962)

Event-Based Simulation

Event Graphs (Schruben 1983)

Event Scheduling

Future Events List

Processing Network Simulation

Activities

GPSS (1961),
SIMAN/Arena,
ProModel,
ExtendSim, Simio,
AnyLogic, ...

Agent-Based Simulation

! ! !

PART I

DISCRETE DYNAMIC SYSTEMS

WHAT IS A DISCRETE DYNAMIC SYSTEM (DDS)?

A real world system consisting of **objects** and a discrete flow of **events** such that at any moment in time, the system's past is a sequence of situations each characterized by

1. a time point *t* (the situation time)
2. the ***system state*** at *t* (as a combination of the states of all objects of the system), and
3. a set of ***imminent events***, to occur at times greater than *t*.

and each situation S_{t+1} is created from S_t via ***causal regularities*** triggered through the events occuring at *t*.

CAUSAL REGULARITIES

An event $e@t$ causes:

1. ***state changes*** Δ of affected objects, and
2. ***follow-up events*** $e_1@t_1$, $e_2@t_2$,...

according to the ***dispositions*** of affected objects, which can be generalized as ***causal regularities*** of the form

$t, O, e@t \rightarrow \Delta, \{e_1@t_1, e_2@t_2, \ldots\}$ with $t_i > t$

with O being the set of the system's object states at time t, such that

$O' = \mathrm{Upd}(O, \Delta)$

is the resulting changed system state.

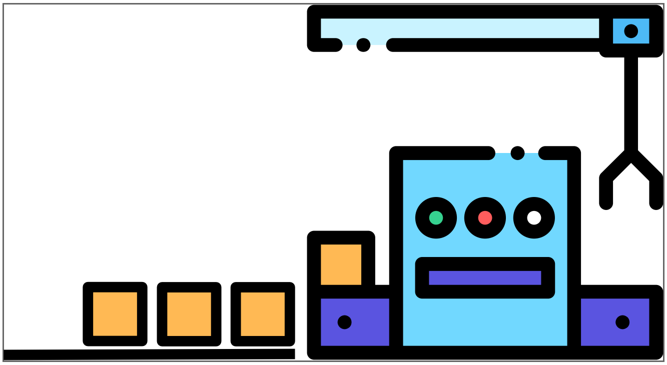
MODELING A DDS

Computationally, a DDS can be represented by an *Object Event Model (OEM)* consisting of:

1. ***object types*** *OT*, e.g., in the form of *classes* of an object-oriented language;
2. ***event types*** *ET*, e.g., in the form of *classes* of an object-oriented language;
3. ***event rules*** *R* representing *causal regularities*, e.g., in the form of `onEvent` methods of the class that implements the triggering event type.

While *OT* and *ET* can be defined by a UML Class Diagram, the set of event rules *R* can be defined by an Extended Event Graph (or a basic DPMN Process Diagram).

EXAMPLE: A MANUFACTURING WORKSTATION

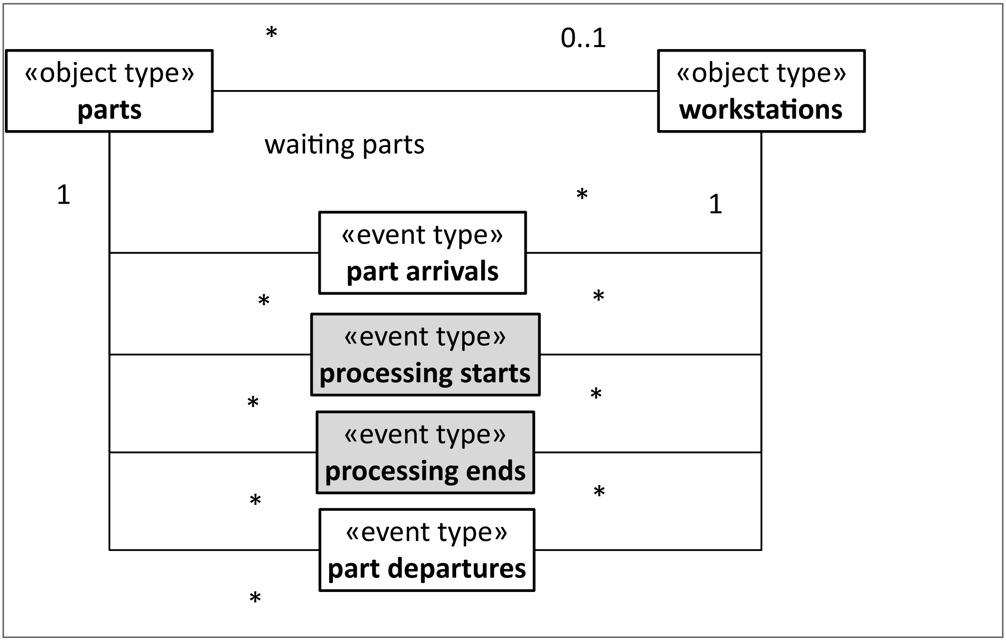


- Event rule 1: When a new ***part arrives*** at the workstation it is added to its input buffer, and if the workstation is available, processing starts.
- Event rule 2: When ***processing starts***, the next part is fetched from the input buffer and is being processed until processing ends.
- Event rule 3: When ***processing ends***, the processed part is removed, and, if the input buffer is not empty, the workstation fetches the next part and starts processing it.

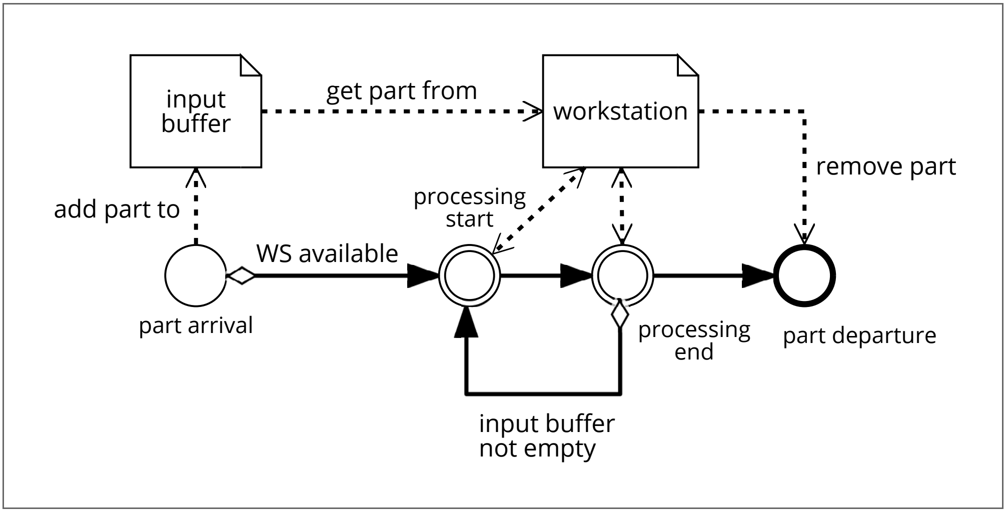
Potentially relevant ***object types***: parts, workstations.

Potentially relevant ***event types***: part arrivals, processing starts, processing ends, part departures.

CONCEPTUAL INFORMATION MODEL



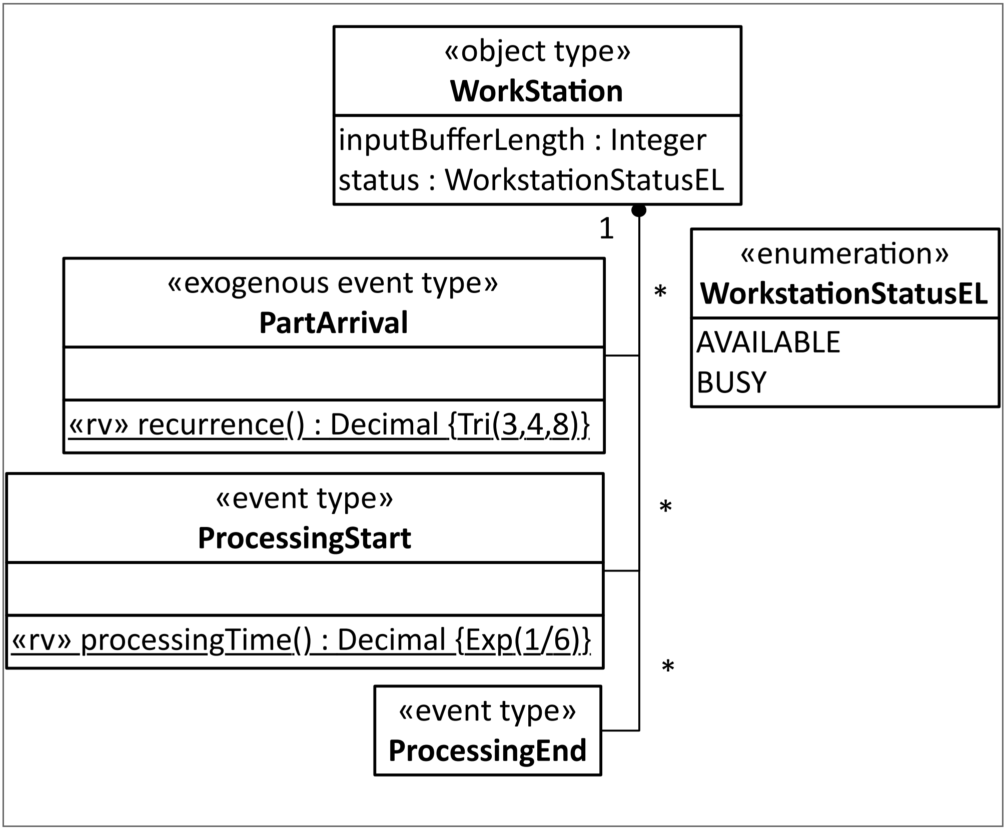
CONCEPTUAL PROCESS MODEL



PART II

INFORMATION MODELING WITH UML CLASS DIAGRAMS

DEFINING OBJECT AND EVENT TYPES



«object type»
WorkStation

inputBufferLength : Integer
status : WorkstationStatusEL

«enumeration»
WorkstationStatusEL

AVAILABLE
BUSY

«exogenous event type»
PartArrival

«rv» recurrence() : Decimal {Tri(3,4,8)}

«event type»
ProcessingStart

«rv» processingTime() : Decimal {Exp(1/6)}

«event type»
ProcessingEnd

PART III

PROCESS MODELING

WITH

EXTENDED EVENT GRAPHS

EVENT GRAPHS

Event Graphs (EGs) have been proposed for DES modeling by Schruben in 1983.

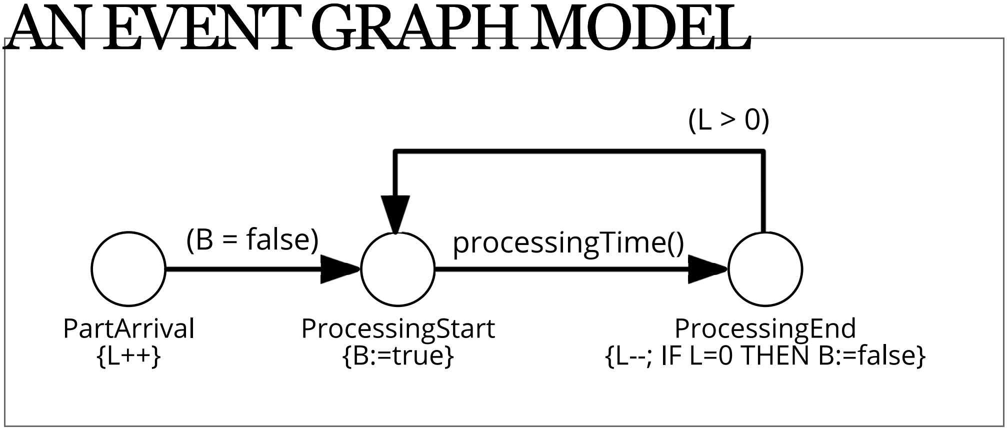
Strengths:

- EGs provide an intuitive visual modeling language.
- EGs capture the fundamental *event scheduling* paradigm.

Weaknesses:

1. EGs lack a visual notation for (conditional and parallel) branching.
2. EGs do not support OO state structure modeling (with objects/classes and attributes).
3. EGs do not allow combining events and activities.

AN EVENT GRAPH MODEL



The integer variable *L* denotes the length of the input buffer.
The Boolean variable *B* denotes the busy/available status of the machine.

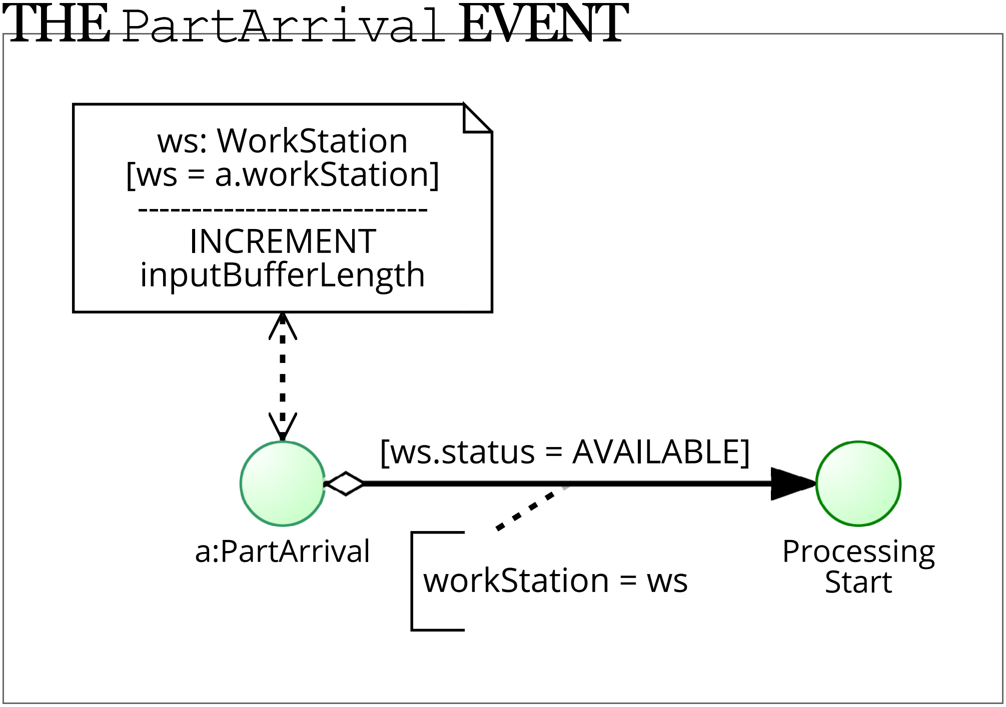
DPMN

...is the *Discrete Event Process Modeling Notation*, which extends *Event Graphs* by adding:

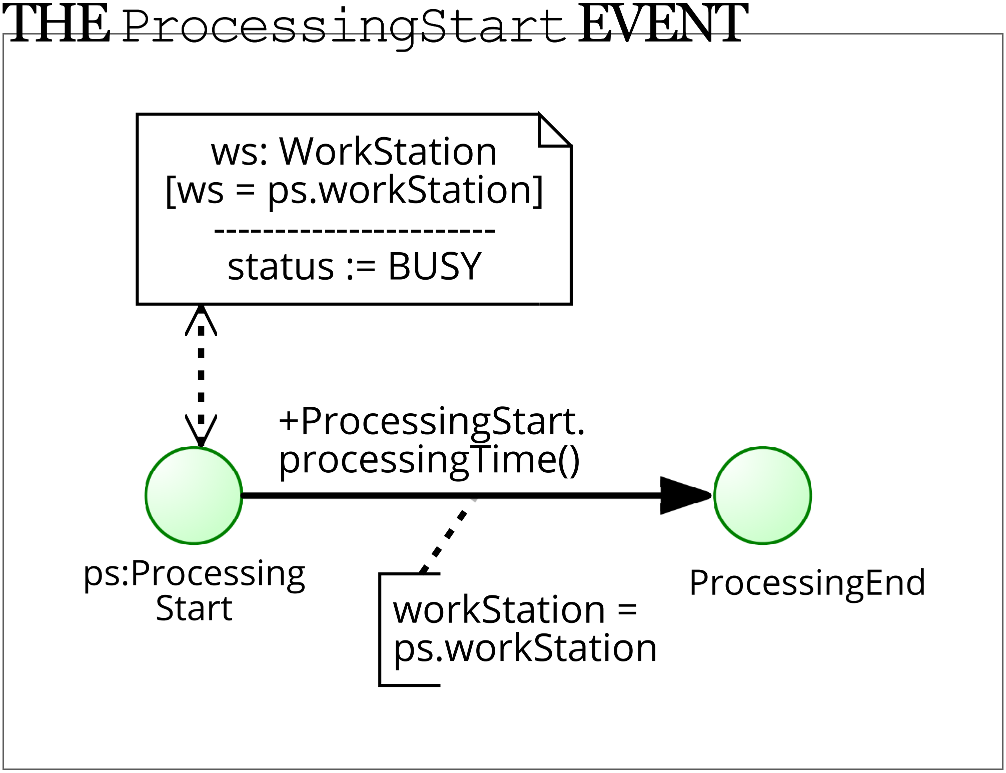
1. *Exclusive/Inclusive/Parallel* ***Gateways*** for conditional/parallel branching
2. ***Data Objects*** for replacing "state variables" (like *L*) with attributes (like *WorkStation::inputBufferLength*)

A *DPMN Process Model* is composed of ***Event Rule Models***.

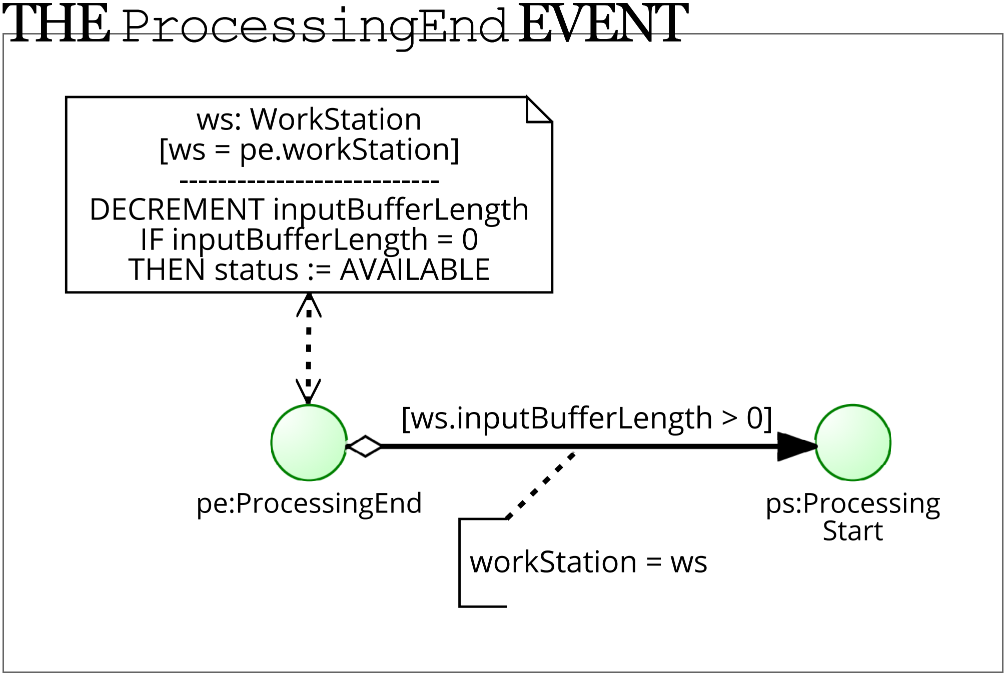
AN EVENT RULE MODEL FOR THE `PartArrival` EVENT



AN EVENT RULE MODEL FOR THE `ProcessingStart` EVENT



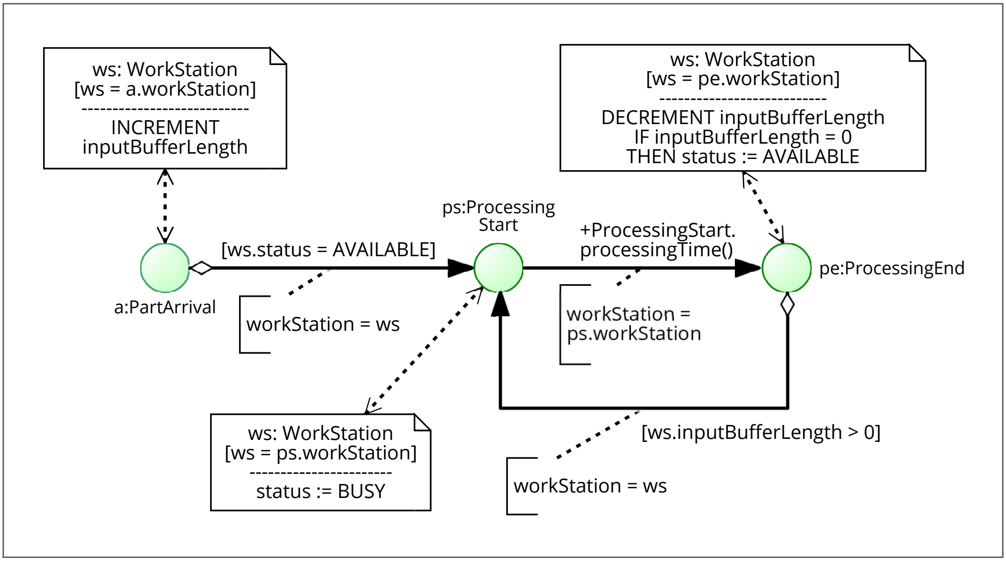
AN EVENT RULE MODEL FOR THE `ProcessingEnd` EVENT



ws: WorkStation
[ws = pe.workStation]

DECREMENT inputBufferLength
IF inputBufferLength = 0
THEN status := AVAILABLE

A DPMN MODEL OF THE WORKSTATION PROCESS



ws: WorkStation
[ws = a.workStation]

INCREMENT
inputBufferLength

ws: WorkStation
[ws = pe.workStation]

DECREMENT inputBufferLength
IF inputBufferLength = 0
THEN status := AVAILABLE

ps:Processing
Start

[ws.status = AVAILABLE]

+ProcessingStart.
processingTime()

pe:ProcessingEnd

a:PartArrival

workStation = ws

workStation =
ps.workStation

ws: WorkStation
[ws = ps.workStation]
----
status := BUSY

[ws.inputBufferLength > 0]

workStation = ws

PART IV

SIMULATION WITH OESjs

AN OESjs MODEL OF THE WORKSTATION PROCESS

... consists of

- an object type definition: `Workstation`
- three event type definitions: `PartArrival`, `ProcessingStart` and `ProcessingEnd`

It can be run as an online simulation at **https://sim4edu.com/sims/102**.

Workstation.js

```
var WorkStation = new cLASS({
  Name: "WorkStation",
  supertypeName: "oBJECT",
  properties: {
    "inputBufferLength": {range: "NonNegativeInteger",
        label: "Input buffer length"},
    "status": {range: WorkstationStatusEL, label: "Status"}
  }
});
```

PartArrival.js

```
var PartArrival = new cLASS({
  Name: "PartArrival",
  supertypeName: "eVENT",
  properties: {
    "workStation": {range: "WorkStation", label:"Workstation"}
  },
  methods: {
    "onEvent": function () {
      var events=[], ws = this.workStation;
      // add part to buffer
      ws.inputBufferLength++;
      // update statistics
      sim.stat.arrivedParts++;
      // if the work station is available
      if (ws.status === WorkstationStatusEL.AVAILABLE) {
        // schedule the part's processing start event
        events.push( new ProcessingStart({ workStation: ws}));
      }
      return events;
    }
  }
});
```

CONCLUSION AND OUTLOOK

- OES is a new DES paradigm with a *formal* semantics and an *ontological* foundation.
- The preferred modeling languages for OES are *UML Class Diagrams* and *DPMN Process Diagrams*.
- OES has been implemented in *JavaScript*, a *Python* implementation will follow.
- Basic OES can be extended by adding *Activities*, *Processing Networks*, *Agents*, etc.

SEE ALSO

- Gerd Wagner: **An Abstract State Machine Semantics For Discrete Event Simulation**, *Proc. of the 2017 Winter Simulation Conference*.
- Gerd Wagner: **Information and Process Modeling for Simulation – Part I: Objects and Events**. *Journal of Simulation Engineering* 1:1, 2018.
- Gerd Wagner: **Information and Process Modeling for Simulation – Part II: Activities and Processing Networks**. 2019.
- Available on `dpmn.info`