Brandenburgische Technische Universität Cottbus
Institut für Informatik
Lehrstuhl Internet-Technologie

Master Thesis

Object Event Simulation of Activity Networks with Java and AnyLogic

Yevhenii Samorodov
MatrikelNr.: 3628572
Informatik

30.01.2021
07.10.2021

Thesis Supervisor: Prof. Dr. Gerd Wagner
Chair of Internet Technology Brandenburg University of Technology Cottbus

Reviewers:
Professor Dr. Gerd Wagner
Professor Dr. Andriy Panchenko

Table of Contents

1. Introduction

Simulation modeling is widely used in the research and management of complex discrete systems and the processes occurring in them. Such systems include economic and production facilities, airports, social dynamics, health care, oil pumping complexes, seaports, irrigation systems, computer networks and many others. The widespread use of simulation modeling is due to the fact that sometimes it is expensive or impossible to experiment on a real object, or it is necessary to simulate the behavior of a system in time. Growing interest in the field of simulation modeling has pushed forward the development of simulation modeling software, which was originally created independently of each other and was adapted to the specifics of different subject areas and research methods. Each of these modeling software applications has its own fundamental modeling language underneath, which implements a concrete simulation paradigm (or even several). This Master Thesis provides an overview of how a simulation framework designed in JavaScript may be ported to Java, reconstructs Object Event Simulation models in AnyLogic and Java, and describes the mapping rule between them.

This thesis starts with a description of Object Event Simulation (OES) and investigation of case study models in Chapter 2. Following that, Chapter 3 discusses the outcomes of porting the OES framework from JavaScript to the Java programming language, emphasizing the tools used and the difficulties encountered. After this chapter, the AnyLogic modeling environment gets introduced and used for model reconstruction in Chapter 4. The following Chapter 5, introduces a mapping rules and transformation ideas between the OES meta-model and the AnyLogic meta-model. Finally, Chapter 6 concludes the work and gives some outlook on possible future work.

1.1. Related works

As will be seen later in this thesis, a significant number of references to other academic papers have been included. However, in this section, two works will be highlighted that have had an impact on the fundamental elements of this thesis and will be discussed in detail.

1.1.1. Introduction to simulation using JavaScript

The tutorial by G. Wagner [1] demonstrates how JavaScript may be used to create web-based simulations. It has a two-fold structure, first introducing JavaScript programming using the topic of simulation, and then describing a simulation approach using the programming language JavaScript. Additionally, it provides guidance on how to create a

Simulation Design Model, which could be used for implementation with JavaScript or other object-oriented (OO) languages.

This paper is a good starting point for understanding how DES simulations can be step-wise implemented in the OO language. A developer can study the logic of simulation model development, investigate design techniques for necessary simulation components, and evaluate necessary semantic constructions in JavaScript (by code snippets) and attempt to integrate them into the target programming language.

1.1.2. Ontologies and simulation: a practical approach

An ontology is an explicit specification of a conceptualization [2]. Conceptualization is not only needed for robots and agents, but also for human designers, e.g. of database or application domains of software projects. Software developers normally derive the needed domain knowledge from requirements, sometimes from models of previous projects. Analysis and design for software projects might very well profit from existing ontologies covering their application domain.

In this paper, the authors demonstrate how to successfully apply ontologies in simulation by using current innovations in systems and software engineering. They used OMG SysML to develop an ontology implementation called a domain-specific language (DSL) for a class of simulation applications; the DSL is used to provide a specially designed (conceptual) user model for a domain problem. After that, the model transformation is applied to automate the transformation to a computational simulation model. During the transformation, a proof-of-concept was constructed using the object-oriented simulation language AnyLogic as the target language.

2. Object Event Simulation (OES)

Object Event Simulation (OES) is a new Discrete Event Simulation (DES) paradigm combining object-oriented modeling and event-based simulation (including event scheduling). An OES design model, providing a computationally complete description of a DES model, consists of an information design model and a process design model. The information design model specifies the types of objects and events that occur in the process design model, while the process design model defines causal regularities involving events as causes and object state changes as well as follow-up events as effects. [4]. OEM refers to a modeling technique that is based on the OE paradigm. This method requires the selection or definition of an information modeling language (for example, Entity Relationship Diagrams or UML Class Diagrams) and a process modeling language (such as UML Activity Diagrams or BPMN Process Diagrams).

The Object Event Simulation subsumes the Event-Based Simulation paradigm with its Event Scheduling and Future Event List features, Processing Network Simulation with activities, and Agent-Based Simulation as a high-level form of Discrete Event Simulation. In this chapter, the basic concepts of each paradigm will be briefly described and two further case studies of OEM/DPMN models presented, summarizing papers by G. Wagner [6], [7], [8], [9].

2.1. Introduction

2.1.1. Discrete Event Simulation (DES)

Discrete-event modeling is a form of simulation modeling. In discrete-event simulation, the functioning of the system is represented as a chronological sequence of events. An event occurs at a certain point in time and changes the overall state of the system. Jeffrey Gordon, the creator of the still-popular software program GPSS, presented this concept in the 1960s [5]. In his work, he proposed to use the concepts of entities, resources and flowcharts.

As discussed in [10], there are three fundamental categories of entities:

1. **Objects**
2. **Tropes**, which are existentially dependent entities such as the qualities and dispositions of objects, as well as their connections with one another
3. **Events**

A real-world discrete dynamic system (or discrete event system) consists of:

- **Objects** (of various types) having a state (consisting of qualities), which may be changed by
- **Events** (of various types) occurring in a sequence of time points, causing state changes of affected objects and follow-up events.

These are the two most fundamental ontological categories, which are also interconnected, since **objects participate in events**.

The system's past is made up of a series of situations, each of which is defined by:

- a time point t
- the system state at t (as a combination of the states of all objects of the system)
- a set of imminent events that will occur at a time greater than t.

and each situation S_{t+1} is created from S_t by causal regularities caused by events occurring at time t.

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 the affected objects, which can be characterized as causal regularities of form t, O, e@t $\rightarrow \Delta$, $\{e_1@t_1, e_2@t_2,...\}$ with $t_i >$ t with O denoting the set of the system's object states at time t in such a way that O' = Upd(O, Δ) is the resulting changed system state. [4]

For modeling a discrete event system (also using OES) as a state transition system, we have to:

1. Describe its **object and event types**, e.g., in the form of classes of an object-oriented language;
2. Specify for any **event type** which **causal regularity** (disposition types) are responsible for **state changes** of objects and **follow-up events** caused by the occurrence of an event of that type, e.g., in the form of **event rules**.

2.1.2. Event-Based Simulation

There are two fundamental methods of representing the knowledge about a discrete-event system: time-based and event-based simulation. In event-based simulation, the system is viewed as a series of instantaneous events occurring at any point in time. Event-based simulation models a system's state using state variables and event scheduling with a Future Events List, which contains the next events and their occurrence time, to model its dynamics. Only systems for which the points in time at which the events occur can be calculated in advance are suitable for event-based simulation.

Event Graphs are a way of representing the Future Event List logic for a discrete-event model. Nodes and directed edges make up an Event Graph. Each node represents a state change or event, and each edge represents the scheduling of subsequent events [11]. For example, an event graph might look like this:

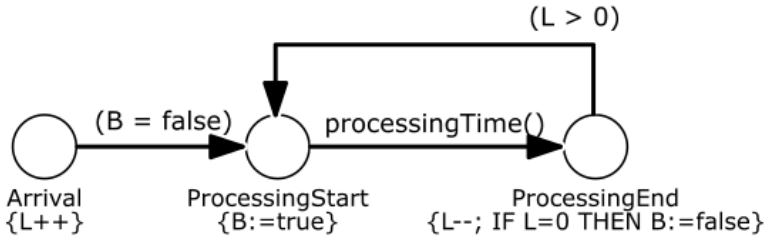


Figure 2.1.: An Event Graph defining an ES model with two state variables and three event types.

L represents the length of an arrival queue, B represents whether a performer is busy or not. There are also three event variables in the form of circles that indicate Arrival, ProcessingStart, and ProcessingEnd events. It also specifies the order in which certain types of events occur, as well as the state changes that they induce, in the form of conditional variable assignments.

2.1.3. Activity-Based Simulation

An activity is a composite event with a duration greater than zero, performed by an agent. An activity is composed of, and temporally framed by, a start and an end event, which are instantaneous (zero-duration) events.

Objects are participating in activities, just as they do in events. An activity in the real world requires at least one participant: the activity's performer. As a result, a conceptual model for each activity type should include the type of objects that serve the performer function. Alternatively, we may model an activity without modeling any participants in a simulation design model by leaving the performer of an activity implicit.

Furthermore, activity may necessitate additional resources, and distributing the necessary resources from resource pools during the course of a discrete event simulation is critical to keep it running. These activities, which involve one or more participants playing a Resource Role, are also known as *Resource-Constrained* Activities. Resource (Cardinality) Constraints, which describe how many resources must be available to start a given activity, are part of a Resource-Constrained Activity. Typically, each activity node in an Activity Network (AN) has a queue of planned activities ("tasks") that are awaiting the availability of needed resources (in particular, their performers). When an activity is scheduled but cannot begin due to the absence of a needed resource, the planned activity is queued in so called "Task Queue". Only when a successor activity node does not require additional or different resources, it doesn't need a (resource allocation) queue and can be started immediately at the completion of a predecessor activity,

which could be indicated by an event flow arrow.

For instance, a MakePizza activity may require two pizza-makers and an oven. Such resource cardinality constraints are defined at the type level. When defining the activity type MakePizza, these resource cardinality constraints are defined in the form of one mandatory association with the object type PizzaMaker with a multiplicity of 2 and one association with the resource (count) pool Ovens with a multiplicity of 1 ("exactly one"). Then, in a simulation run, a new MakePizza activity can only be started, when two PizzaMaker objects are available and the resource (count) pool Ovens has a non-zero value.

As described in [7], simulator can automatically collect the following statistics for all types of resource-constrained activities:

1. Throughput statistics: the numbers of enqueued and dequeued planned activities, and the numbers of started and completed activities.

2. Queue length statistics (average, maximum, etc.) of its queue of planned activities.

3. Cycle time statistics (average, maximum, etc.), where cycle time is the sum of the waiting time and the activity duration.

4. Resource utilization statistics: the percentage of time each resource object involved is busy with an activity of that type.

Since its start in the 1960s, modeling resource-constrained activities has been a key challenge in the subject of Discrete Event Simulation (DES), yet it has been neglected and is still considered an advanced topic in the field of Business Process Modeling (BPM). While the Business Process Modeling Notation (BPMN) allows one to allocate resources to activities, it doesn't allow one to model resource pools or define resource cardinality restrictions or parallel participation multiplicity requirements. A business process model describes an Activity Network (AN) as a set of event and activity nodes that are connected by event flow arrows and resource-dependent activity scheduling (RDAS) arrows. Event and activity nodes can be associated with objects representing their participants. These participating objects, in the case of an activity node, contain the resource objects required to perform the activity and often a particular resource object representing the activity performer. The participation of objects in events and activities, including the resource roles of activities, as well as resource cardinality constraints, parallel participation constraints, alternative resources, and task priorities are defined in the underlying Object Event (OE) class design model, as well as the types of objects, events and activities participating in the business process.

Gordon (1961) proposed the Seize and Release resource management algorithms in the simulation language GPSS for allocating and de-allocating (releasing) resources in the DES paradigm of Processing Networks. As a result, GPSS has developed a standard modeling pattern for resource-constrained activities known as Seize-Delay-Release, which states that when simulating a resource-constrained activity, its resources are first allocated and then de-allocated (released) after a delay (representing the simulated activity's duration). [7]

2.1.4. Processing Activities and Processing Networks

A Processing Activity is a resource-constrained activity that accepts one or more objects as inputs, processes them in some fashion (perhaps converting them from one type of object to another type), and produces one or more objects as outputs at a processing station. The processed objects were referred to as "transactions" and "entities" in GPSS and SIMAN/Arena respectively, while in DPMN, they were referred to as "Processing Objects".

As illustrated in the Figure 2.2, one or more objects are engaged in an activity ontologically. They represent resources in certain cases and processing objects in others.

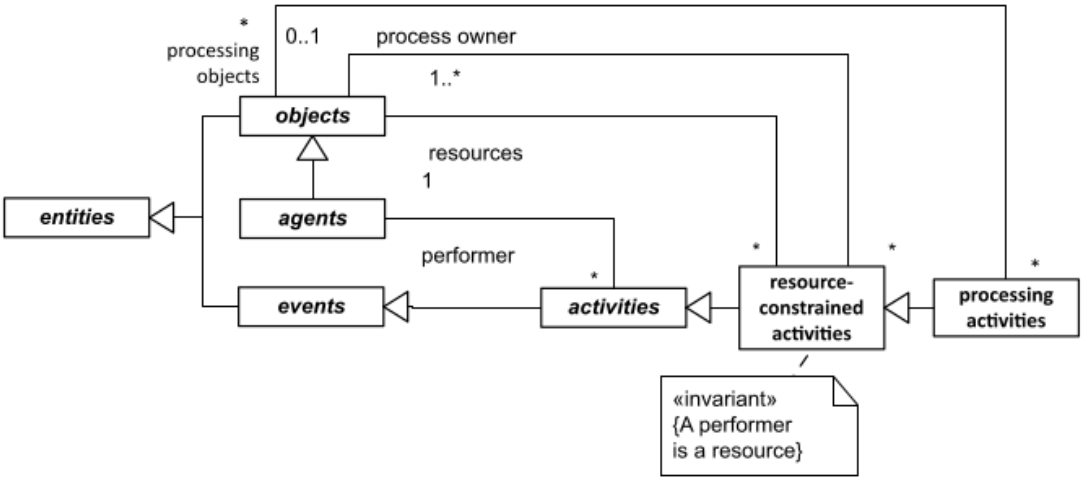


Figure 2.2.: The concept of Processing Activities. Source: [8]

Processing activities generally involve fixed physical resources, such as rooms or workstation machines, which form the Processing Network's (PN) inner nodes. In a PN model, each node represents an object as well as a event type. There are three types of processing nodes:

- An **entry node**, which consists of an entry station object (e.g., a reception area or a factory entrance) and an arrival event type for arriving processing objects (such as people or manufacturing parts).
- A **processing node**, made up of two components: a processing station object (which is commonly used as a resource object, such as a workstation or a room) and a processing activity type.
- An **exit node**, which consists of an exit station object and a departure event type.

So a Processing Object enters such a network through an Arrival event at an Entry Node, is then routed down a chain of Processing Nodes where Processing Activities are performed, and ultimately leaves the network via a Departure event at an Exit Node.

In a PN, all processing nodes have an input buffer (or queue) filled with processing objects that wait to be processed and a task queue (or a queue of planned activities, similar to AN) waiting for the availability of the required resources.

In addition to the Activity Network statistics given in Section 2.1.3, a simulator can automatically gather the following statistics for PNs:

1. The total number of processing objects that arrived and departed the system.
2. The number of processing objects currently in use (either in a queue/buffer or being processed).
3. The amount of time that a processing object spends in the system on average (also called throughput time).

2.1.5. ANs versus PNs

Business process models created using BPMN are Activity Networks (ANs), whereas business process models created with traditional DES tools such as Simio and AnyLogic are Processing Networks (PNs).

A PN's nodes describe locations in a network space, which might be two-dimensional or three-dimensional Euclidean. As a result, OE Processing Network models are spatial simulation models, whereas basic OE and OE activities models permit for space abstraction. When processing objects are routed to a subsequent processing activity, they are moved to the next processing node's position. The underlying space model enables a realistic visualization of a PN simulation with processing items as moving objects. To completely conclude, it can be stated that AN models differ from PN models because they abstract from space: nodes in an AN do not have a physical location, but nodes in a PN do have a physical place. There are (implicit) task queues in ANs with resource-constrained activities, which (usually) have an infinite capacity. Due to space constraints, the processing nodes of a PN often have input buffers (queues) with a limited capacity.

2.2. Case studies

In this chapter, two case studies of OEM/DPMN models are presented. It will give an impression of how to start the information modeling process, in what order to specify essential participants of the simulation model, and how to receive the computationally complete specification as the end result.

2.2.1. Make and Deliver Pizza model

2.2.1.1. Conceptual Information Model

A pizza service company has three consecutive activities:

1. Take order with the help of *order takers*;

2. Make pizza with the help of *pizza makers* and *ovens*;

3. Deliver pizza with the help of a crew of pizza delivery *scooter drivers*.

Additionally, customers may become frustrated and hang up the phone without making an order when order takers are unable to deal with the volume of incoming calls.

Given the above description, we may define following resource pools for the model:

- pizza makers
- pizza ovens
- delivery staff
- scooters

Note, that while only an order taker is responsible for Take Order activity, Make Pizza activity requires the use of both an oven and two pizza makers.

As mentioned in 2.1.1, for the modeling of DES systems we need to describe object and event types of the system. The potentially relevant object types of the model given are:

- pizza service company,
- customers,
- orders,
- pizzas,
- order takers,
- pizza makers,
- pizza ovens
- delivery scooter drivers,
- scooters.

Potentially relevant types of events and activities are:

- pizza ordering calls coming in from customers,
- order taking (an activity performed by order takers),
- customers hanging up the phone when having to wait for too long,
- pizza making (performed by pizza makers using ovens),
- pizza delivery (performed by delivery staff using scooters).

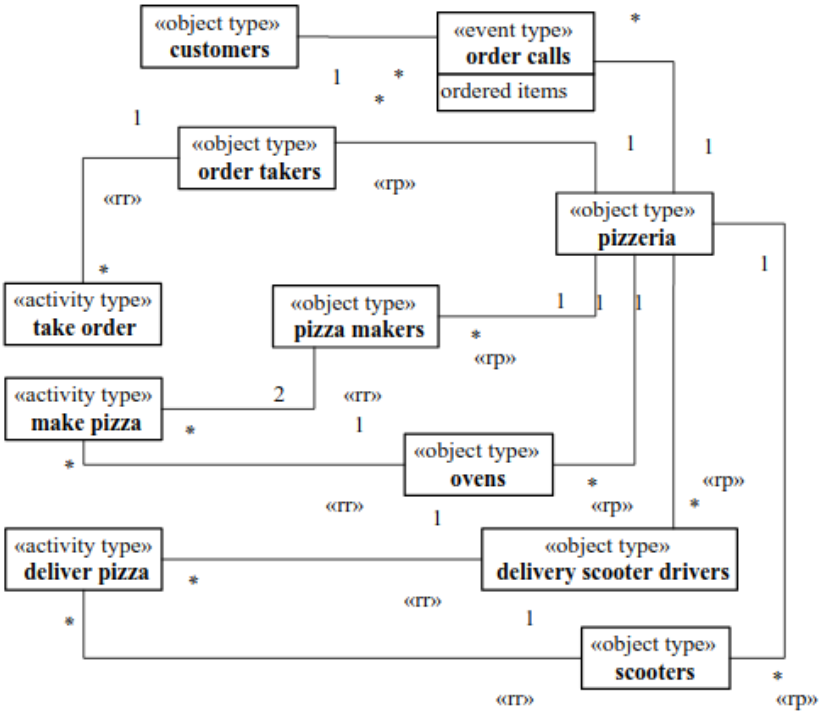


Figure 2.3.: The OE digram of the Make and Deliver Pizza model. Source: [6]

The types of object, event, and activity, as well as their participation relationships, can be graphically expressed in a conceptual information model using a particular type of UML class diagram known as the Object Event (OE) class diagram, as illustrated at Figure 2.3

The annotations ”rr” and ”rp” at the end of an association denote resource roles (like ”oven”) and resource pools (like ”ovens”), respectively. The numbers stand for the cardinality constraints of resource roles. For instance, a Make Pizza activity requires 2 pizza makers and 1 oven.

2.2.1.2. Conceptual Process Model

The BPMN diagram below depicts a conceptual process model of the Make-and-Deliver-Pizza business process, with three swimlanes representing the three process performers:

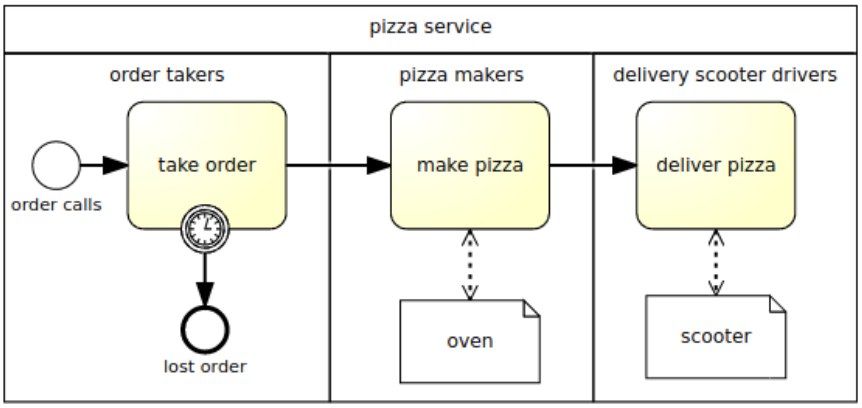


Figure 2.4.: The BPMN diagramm of the Make and Deliver Pizza model. Source: [6]

The BPMN model has three issues, which could be solved in DPMN, namely:

1. **Modeling of reneging behavior**. The BPMN Boundary Timeout Event circle is supposed to model the reneging behavior of waiting customers, who lose their patience and hang up the phone without placing an order. It applies to the entire cycle time of take order activities, which means that already started activities may also be canceled.

2. **Insufficient support of resource management**. While BPMN provides the modeling of activity performers using swimlanes (organizational roles with related resource pools), it does not provide the modeling of other types of resource objects. As a workaround for the given diagram (2.4), BPMN Data Objects were created for the resource pools ovens and scooters.

3. **Uniformity**. The usage of ”sequence flow” arrows for sequencing both events and activities is standard (and semantically overloaded) in the BPMN paradigm. While incoming ”sequence flow” arrows do not mean that an activity is started, but rather that a new task (planned activity) is enqueued (and only started when all required resources are available), the incoming ”sequence flow” arrow in the case of the lost order event means that a new event is scheduled to occur immediately.

These three BPMN problems have been addressed in DPMN, which distinguishes resource-dependent activity start arrows from event scheduling arrows, as seen in the DPMN process diagram below:

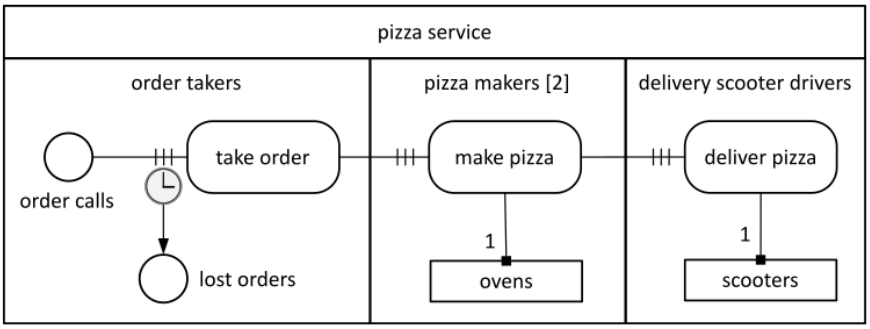


Figure 2.5.: The DPMN diagramm of the Make and Deliver Pizza model. Source: [6]

The timeout event circle is attached to the three bars of the resource-dependent activity scheduling arrow. This implies that the reneging timeout applies to the waiting phase, and not to the order taking activity as a whole.

There are two types of arrows in the DPMN diagram. Event Scheduling arrows were proposed by Schruben (1983) for Event Graphs and are not part of BPMN. The Resource-Dependent Activity Scheduling arrow denotes the addition of a new scheduled activity to the task queue of the next activity.

The types of objects, events, and activities in DPMN models are defined by an underlying Object Event (OE) Class Model. DPMN may be used to create both conceptual process models (models of real-world processes) and process design models (models of computational design artifacts). Design models are created on the basis of conceptual models in model-based simulation engineering. In the framework of the Object Event Modeling and Simulation (OEM&S) paradigm, DPMN was created as an extension of Event Graphs by including features from BPMN.

It is possible to present the underlying OE class model's computational features in an enhanced DPMN process diagram. In particular, the performer roles of activity types, such order taker and pizza maker in swimlanes. The organization under consideration has a resource pool for each organizational role defined in the underlying OE class model.

2.2.1.3. Simulation design

The following simplifications has been made in the simulation design:

- only investigate one unique pizza service provider which must not be modeled as an explicit object.
- abstract away from individual customers, orders and pizzas.
- combine the resource roles of delivery scooter driver and scooter, leaving just scooters as deliver pizza activity resources.

An information design model (Figure 2.6), represented in the form of an OE class diagram, is created from a conceptual information model by removing elements that are not

design-relevant and improving object, event, and activity classes by adding attributes, functions, and methods. All resource object types, such as OrderTaker and Oven, now have a status attribute, and all activity classes now have a class-level duration function.

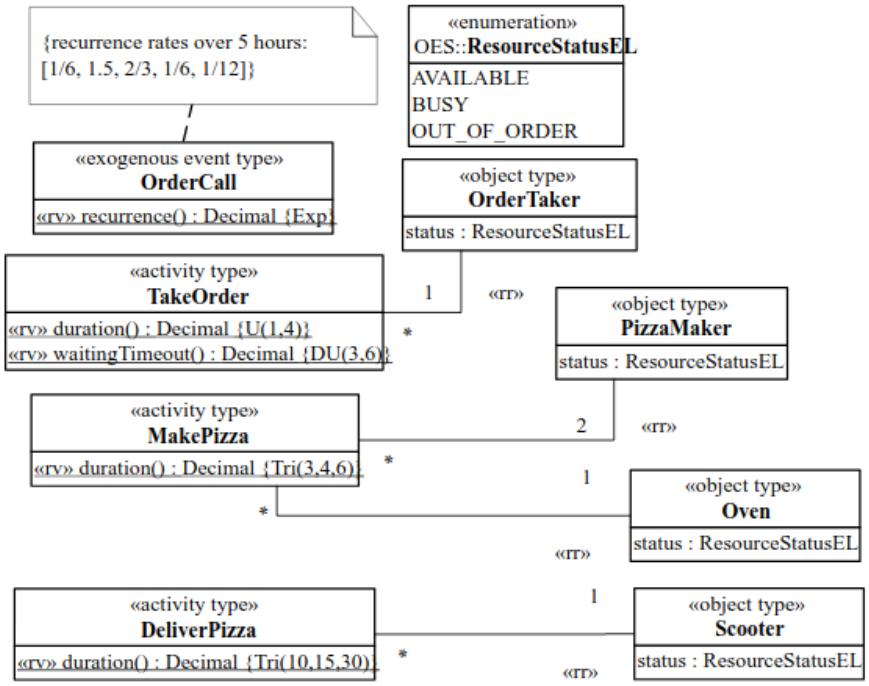


Figure 2.6.: An information design model that defines the types of objects, events, and activities. Source: [6]

You will notice that functions representing random variables, such as activity duration, are tagged with the UML stereotype ”rv” (meaning ”random variable”). For example, Tri(30,40,50) stands for a triangular probability distribution function (PDF) with bottom and upper limits of 30 and 50, and a median of 40, whereas DU(1,4) stands for a discrete uniform PDF with lower and upper bounds of 3 and 6. An exponential PDF with five different event rates for the five consecutive hours during which the pizza service works is used for the event type OrderCall.

The implicit resource role orderTaker (with a resource cardinality requirement of ”exactly 1”) links the activity type TakeOrder to the object type OrderTaker. A resource role allocates resource objects to tasks. Similarly, MakePizza is linked to PizzaMaker and Oven via the (implicitly named) resource roles pizzaMakers (exactly 2) and oven (exactly 1).

Resource roles (like pizzaMakers), role types (like PizzaMaker) and cardinality restrictions (like ”exactly 2”) are defined via an OE class design diagram. Resource role types and resource pools normally match in an OE simulation. A resource pool’s name is the same as the resource role type’s name, but pluralized and beginning with a lowercase

character. The resource pool for PizzaMaker, for example, is pizzaMakers. For each of the five hours of operation of the pizza service company, OrderCall events are exogenous (in the attached invariant box).

2.2.1.4. Process design model

A process design model, illustrated below (Figure 2.7) in the form of a DPMN process diagram, is constructed from a conceptual process model by:

- Abstracting away non-design-relevant elements.
- If necessary, introducing event variables.
- Specifying state changes of objects influenced by events using object variables defined in the form of Data Object boxes.
- Formalizing decision criteria using event and object variables as inputs.

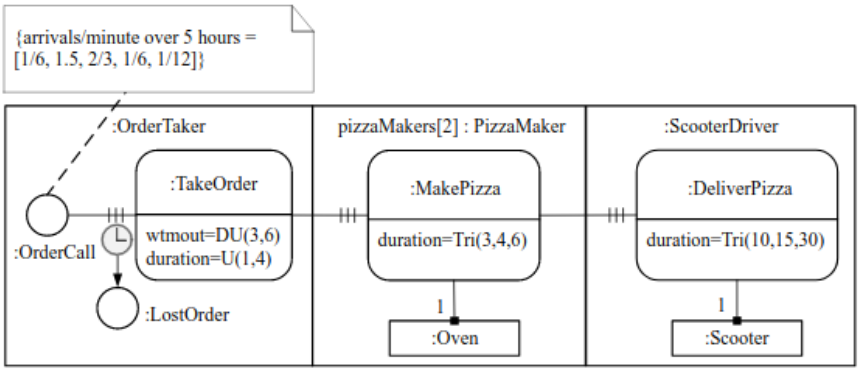


Figure 2.7.: A computationally complete process design for the Make and Deliver Pizza business process. Source: [6]

2.2.2. The Load-Haul-Dump Model

A haul service company takes orders for the transportation of huge quantities of soil from a loading site to a dump site via dump trucks and wheel loaders. There are two resource pools at the company: one for trucks and another for wheel loaders.

2.2.2.1. Conceptual Information Model

The potentially relevant object types are:

- haul service company,
- wheel loaders,

- dump trucks.

Potentially relevant types of events and activities are:

- haul requests coming in from customers,
- going to the loading site (an activity performed by trucks and by wheel loaders),
- loading (performed by wheel loaders using trucks as resources),
- hauling (performed by trucks),
- dumping (performed by trucks),
- going back to loading site (performed by trucks),
- going home when the job is done (performed by trucks and by wheel loaders).

Similar to the approach in chapter 2.1, let us graphically represent our information model in a form of Object Event (OE) class diagram (Figure 2.8).

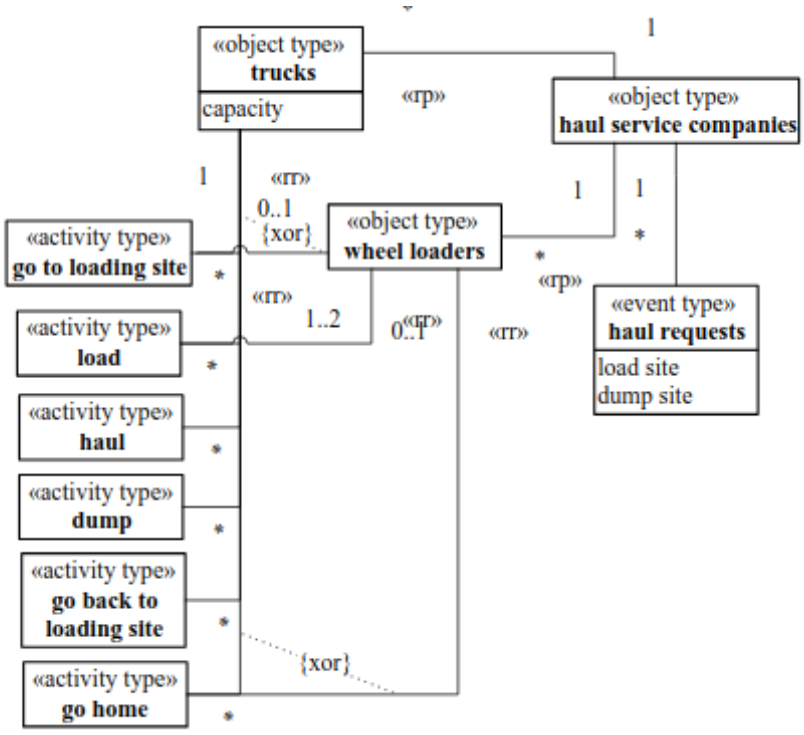


Figure 2.8.: A conceptual OE class model describing object, event and activity types. Source: [6]

The alternative association constraint represented with a dashed line annotated with xor indicates that the activity types ”go to loading site” and ”go home” have either a

resource role truck or a resource role wheel loader. For assigning at least one and up to two wheel loaders to any load activity, the activity type load has both a resource role truck and a resource role wheel loaders.

2.2.2.2. Conceptual Process Model

As illustrated in the DPMN process diagram below, the various types of events and activities may be linked together using resource-dependent activity start arrows and event scheduling arrows:

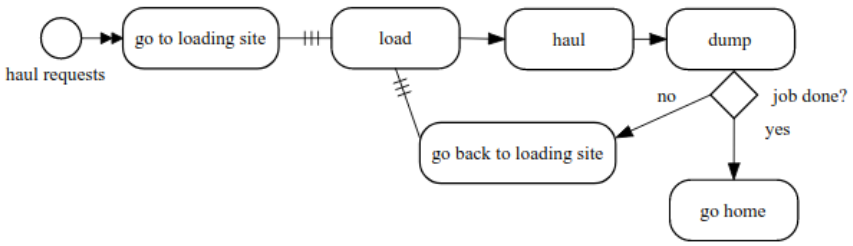


Figure 2.9.: The DPMN diagramm of the Haul and Dump model. Source: [6]

After receiving a haul request, the haul service company sends numerous trucks and wheel loaders to the loading site, as shown by the double arrow between the haul requests event circle and the go to loading site activity rectangle. The resource-dependent activity start arrow from the go to loading site activity shape to the load activity shape indicates that each of these activities enqueues a new scheduled load activity. When the needed resources become available, the enqueued activity will be dequeued and launched. So, as soon as a wheel loader is available, the next load activity will start. Following a load activity, a haul activity and subsequently a dump activity will begin immediately, as indicated by the event scheduling arrows.

2.2.2.3. Simulation design

We examine only one specific haul service company in our simulation design, which does not require detailed modeling. Additionally, we ignore the fact that wheel loaders must drive to and from the loading location by assuming they are already there when the dump trucks arrive.

In the information design model, we must specify a status attribute for each resource object type, such as Truck and WheelLoader, as well as a duration function for each activity type, which generally represents a random variable:

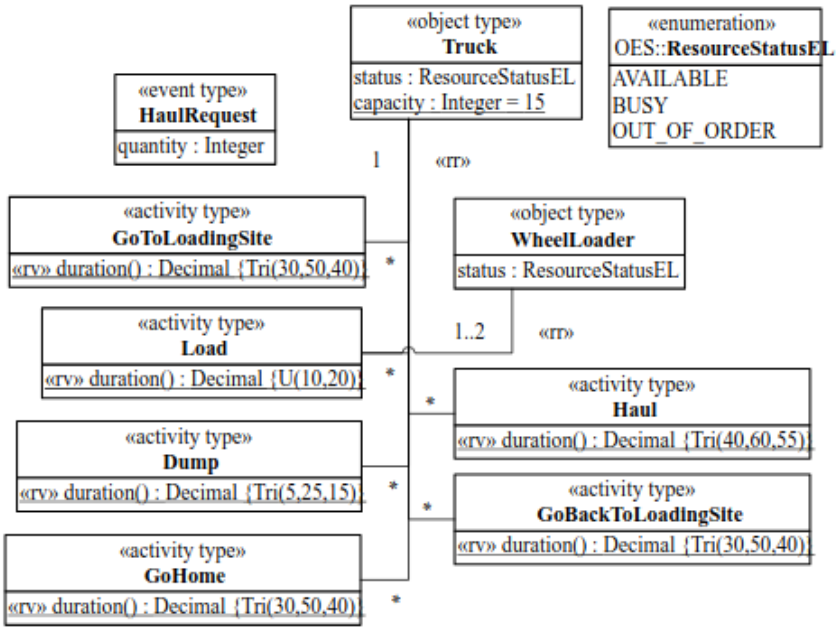


Figure 2.10.: An information design model defining object, event and activity types. Source: [6]

2.2.2.4. Process design model

A process design model, depicted below in the form of a DPMN process diagram, is created from a conceptual process model using the same technique as described in section 2.2.1.4 (Abstracting away from items that are not design-relevant, defining event and object variables, etc)

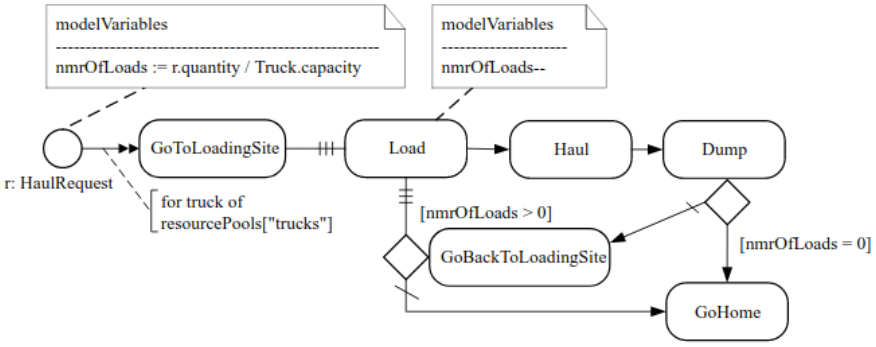


Figure 2.11.: A computationally complete process design for the Load-Haul-Dump business process. Source: [6]

3. Porting the Object Event Simulation Framework from JavaScript (OESjs) to Java

In the practical part of this work, the existing JavaScript implementation of Object Event Simulation (OES) [3] (also designated as "OESjs") had to be ported to the Java programming language. Porting from one programming language to another is a good approach to evaluating the software portability of a particular product/framework/library. The more highly portable software is built, the more it will be able to adapt to changes and continue to be used. In the same time, porting of the software from one language to another is a challenging task to compare the availability of existing libraries and operating system functions during implementation in a target language.

In terms of modeling and simulation, the implementation of the given ontological model in different host languages allows testing its well-formedness and proper behavior.

3.1. Java as a language for simulation model development

Java is a programming language and computing platform first released by Sun Microsystems in 1995 [12]. Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This code can be executed on any system that implements the Java Virtual Machine [13], therefore increasing the portability of a software across different systems.

In case of a simulation modeling development, the main advantage of Java language is its extensibility. Java "libraries" build language extension that are well-suited for simulation modeling requirements, such as:

- **Visualization of a model**. Visualizations are used to understand, enhance and explore the modeling and simulation process and simulation results [14]. There are a variety of open-source visualization libraries available, starting from simple diagrams of data (e.g. Graphviz [15]) to completely immersive and interactive 3D environments (Java3D API [16]).
- **Database connection**. To connect to a database in Java application, one can simply use the Java Database Connectivity (JDBC) API. It supports connectivity to such popular databases as MySQL, PostgreSQL etc.
- **Statistical computing**. Statistical analysis is an essential part of simulation modeling. Application of statistical functions on simulation outcomes allows forecasting of particular events in the system, adjust input parameters or conclude

important results for a business process. Java has a good amount of tools to perform statistical analysis, moreover some Machine Learning Java-based libraries for statistical analysis has been released in the past decade, such as Java Statistical Analysis Tool (JSAT) [17].

- **External software connectivity**. Java has interfaces to other languages like Maple or Wolfram, therefore extending all existing and future Java classes with a new environment and its functionality.

There is already a lot of Java-based simulation software on the market, such as DESMO-J, Ptolemy Project, AnyLogic [18].

3.2. Tools used for development

3.2.1. Spring Boot Framework

Spring came into being in 2003 as a response to the complexity of the early J2EE specifications [19]. It was difficult to create apps using pure Java EE, but with the advent of Spring it became much easier to develop Java programs. Also it allowed to reduce significant amount of a code with the use of Dependency Injection and Common Annotations. For example, Spring JDBC has a bunch of JDBC-packages and templates, which allows easily to separate JDBC from all other classes and maintain transactions in a simple way, unlike standard Java EE approach, where the JDBC was instantiated inside each caller class and transaction manager control was cluttering the code.

Spring Boot is basically an extension of the Spring framework, which allows creating stand-alone, production-grade Spring-based applications [20] with even more reduced amount of code. Spring Boot also provides embedded servers (Tomcat, Jetty) and auto-configuration, therefore giving the opportunity to develop RESTful API.

These features, together with others, make Spring one of the most popular Java and Java EE frameworks nowadays. As mentioned in survey [21] from 2020, nearly half of 2000 respondents are using Spring boot as a server-side web framework at their work.

3.2.2. PostgreSQL

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of PostgreSQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley and has more than 30 years of active development on the core platform.[22]

This database system was chosen for the OES Java Tech Stack because of its extensibility, variety of data types, and good compatibility with other components.

3.2.3. Docker

Docker is an open platform for developing, shipping, and running applications. Docker provides the ability to package and run an application in a loosely isolated environment

called a container. Containers are lightweight and contain everything needed to run the application, so it does not need to rely on what is currently installed on the host. Containers can be shared across different systems and developers can be sure that everyone gets the same container that works in the same way. [23]

In case of OES Java implementation, Docker makes possible to independently run a PostgreSQL instance from an existing image. It allows to immediately start working with DB without manual creation of required tables, schemes, users etc.

The docker script, which is used in OESjava to start a PSQL database container, you can find in the appendix (section A.1).

3.2.4. Thymeleaf

Thymeleaf is a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text. To achieve this, it builds on the concept of Natural Templates to inject its logic into template files.[24]

Thymeleaf makes possible to provide minimal and representative overview of OES application's output and maintain interaction between user and API with the usage of flexible HTML templates.

3.2.5. Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's building, reporting, and documentation from a central piece of information.[25]

Maven is used to connect the Spring dependencies and other external libraries, like Thymeleaf and JUnit.

You can find the structure of the used POM file in the appendix (section A.2)

3.2.6. JUnit

JUnit is a framework for writing tests on the JVM. JUnit provides sufficient methods for testing the entire application, as well as its portions and isolated components. Testing is a very important aspect of porting from one programming language to another, because it allows you to make sure that the implemented logic is correct in the early stages of development and to compare the delivered results of prototype and source code execution.

3.3. Project structure

The Java language allows for decomposition by means of the package construct. Every Java class is part of a package, and packages are hierarchically structured in a package tree. Packages can therefore be considered to be the modules of a Java program [28]. The major packages of the OESjava application, as well as the files included inside them, will be explained in detail in this section.

3.3.1. de.oes.core2.lib

Package with a helper classes. Contains:

- **EventList.java** EventList maintains an ordered list of events
- **MathLib.java** Math/Statistics Library for OES.
- **Rand.java** Randomization library for OES. Provides different probability distribution and randomization functions.
- **SimulatorLogs.java** A utility for logging simulation steps in the console.

3.3.2. de.oes.core2.foundations

Package with foundation objects and classes of OES core. Contains:

- **eVENT.java** An abstract class, which must be extended for all event types of the system under investigation. An OES event may be instantaneous or it may have a non-zero duration.
- **oBJECT.java** An abstract class, which must be extended for all object types of the system under investigation. An OES object has an ID and may have a unique name. If no ID value is provided on creation, an ID value is automatically assigned using the simulation scenarios "idCounter".
- **ExogenousEvent.java** An abstract class, which extends eVENT class with the `reccurence` and `createNextEvent` methods

3.3.3. de.oes.core2.sim

Package with objects and classes required to construct the OE simulation engine. Contains:

- **eXPERIMENTpARAMdEF.java** A complex datatype for experiment parameter definitions
- **eXPERIMENTtYPE.java** An experiment type is defined for a given simulation model and an experiment of that type is run on top of a given simulation scenario for that model.
- **ActivityStat.java** Collects statistic of an activity.
- **GenericStat.java** Collects generic statistic values.
- **Model.java** The model-specific logic has to be defined using instances of this class (such as object types, event types, event routines and other functions for model-specific computations).

- **ReplicationActivityStat.java** The generic statistics per activity type for each replication.
- **ReplicationGenericStat.java** The generic user-defined statistics for each replication.
- **ReplicationStat.java** The resulting statistics composed of the user-defined statistics and the generic statistics (per activity type) for each replication
- **Scenario.java** A specific simulation scenarios for a given model is defined using instances of this class.
- **SimulationStat.java** The resulting statistics composed of the user-defined statistics and the generic statistics (per activity type) for a standalone simulation run
- **Simulator.java** The most essential class, which contains contains all the logic for running the simulation. This class must also be connected/linked to the rest of the participants in the simulation.
- **Time.java** Enumeration to indicate the underlying time mode of a simulation model, which can be either discrete time or continuous time.
- **TimeUnit.java** Enumeration to indicate the underlying simulation time unit of a simulation model, could be one of the time units "ms", "s", "min", "hour", "day", "week", "month" or "year".

3.3.4. de.oes.core2.activities

Package with Activity Network extension of OES core framework. Contains:

- **aCTIVITY.java** An abstract class of composite events that have some duration and typically depend on resources. An activity type is defined as a subtype of the class "aCTIVITY" with an optional class-level "duration" attribute (or function).
- **aCTIVITYsTATE.java** An activity state (of an object) is a set of activity type names.
- **aCTIVITYsTART.java** A pre-defined event type aCTIVITYsTART is used for scheduling activity start events with a constructor parameter "plannedActivity" (of a specific activity type AT), dequeued from AT.tasks. It is an option to assign a duration at activity start time by providing a duration argument to the activity start event constructor. The value of the activityState property of all involved resource objects is updated by adding the activity type name (the activityState is a set of all type names of those activities, in which the object is currently participating).
- **aCTIVITYeND.java** A pre-defined event type that is caused by another simulation event or scheduled if the duration of an activity is known.

- **rANGE.java** Resource role range containing its name, possibly associated resource pool and alternative resource types.
- **rESOURCE.java** Subtype of the class "oBJECT", that is involved in some activities as a resource. Extends the basic "oBJECT" class with resource status, resource pool and involved activity state.
- **rESOURCEpOOL.java** A resource pool can take one of two forms: 1) a count pool abstracts away from individual resources and just maintains an "available" counter of the available resources of some type or 2) an individual pool is a collection of individual resource objects. For any performer role (defined in an activity type definition), an individual pool is defined with a (lower-cased and pluralized) name obtained from the role's range name if it's a position or, otherwise, from the closest position subtyping the role's range.
- **rESOURCErOLE.java** Composite class for resource roles, which allows to link corresponding resource roles and define cardinality constraints.
- **rESOURCEsTATUS.java** Enumeration to indicate the resource status with one of the values "out of order", "out of duty" or "blocked"
- **tASKqUEUE.java** Define a datatype class for queues of tasks (= planned activities)

3.3.5. de.oes.core2.entity

Package with objects (entities), which will be persisted in database. Contains:

- **eXPERIMENTrUN.java** An experiment run is based on a specific scenario.
- **eXPERIMENTsCENARIOrUN.java** Contains input and output parameters of an experiment run.

3.3.6. de.oes.core2.dao

Data Access Object (DAO) is basically an object that provides access to an underlying database.

- **eXPERIMENTrUNDao.java** - used for get or put data to experiment runs DB.
- **eXPERIMENTsCENARIOrUNDao.java** used for get or put data to experiment scenario runs DB.

3.3.7. de.oes.core2.dto

A Data Transfer Object (DTO) is an object that is used to encapsulate data, and send it from simulation server to client. Contains:

- **LogEntryDTO.java** Used to transfer simulation logs.
- **ActivityStatatisticsDTO.java** Used to transfer simulation statistic per activity type.
- **ExperimentsStatisticsDTO.java** Used to transfer simulation statistic of each experiment scenario run.
- **SimulationSettingsDTO.java**Used to transfer simulation scenario settings for each example model.

3.3.8. de.oes.core2.endpoint

An endpoint is a URL pattern used to communicate with an API.

- **MaintainObjectEventSimulationsEndpoint.java** Defines communication channels (RESTful) which are available for users/clients to utilize/connect to via HTTP methods. Triggers execution of a different examination models.

3.3.9. Example models

For implementing the OE class design model for each example with OESjava-Core2, we have to code all object types, event types and activity types specified in the model in the form of Java classes, which are placed under the corresponding package (e.g. "`de.oes.core2.loadhauldump`" for the Load-Haul-Dump model).

For obtaining a complete executable simulation scenario, a simulation model has to be complemented with simulation parameter settings and an initial system state. In general, we may have more than one simulation scenario for a simulation model. For instance, the same model could be used in two different scenarios with different initial states. The simulation scenario are defined as a separate executor service class (under "`de.oes.core2.endpoint.service`" package). You may find an example of the executor service for the Load-Haul-Dump model in the appendix A.6.

A client may invoke an executor class via an API with specified simulation scenario settings. To do this, we must specify an appropriate endpoint within the "MaintainObjectEventSimulationsEndpoint"-Controller. See an example for the Load-Haul-Dump model:

```
@Controller
public class MaintainObjectEventSimulationsEndpoint {

   @Autowired
   private RunLoadHaulDumpSimulationService loadHaulDumpSimulation;

   @GetMapping("/core2/loadhauldump")
   public String setupLoadHaulDump(Model m) {
      m.addAttribute("scenario", new SimulationSettingsDTO());
       return "loadhauldump-settings";
   }
```

```
    @PostMapping("/core2/loadhauldump")
     public String runLoadHaulDump(SimulationSettingsDTO dto, Model model) {
       model.addAttribute("type", dto.getType());
       model.addAttribute("hasLogs", dto.isSimulationLog());
       loadHaulDumpSimulation.run(dto, model);
         return "loadhauldump";
    }
}
```

Listing 3.1: Two endpoints for the Load-Haul-Dump model - one evaluates input parameters, another runs the simulation and returns results

3.4. Main challenges encountered

Java and JavaScript are at the top of the most popular programming languages, and they play a significant role in the IT industry. There are crucial differences between JavaScript and Java, despite the fact that the languages share much of the same syntax. For example, in Java, type safety is tightly enforced, but in JavaScript, it is easily circumvented. Standard library support differs significantly, for example, in how files are handled.

3.4.1. Global variables

A global variable is a variable which can be accessed from any scope. Many programming languages feature declaring a global variable scope, and JavaScript is one of them. In contrast, Java was created as a purely object-oriented programming language and the use of global variables is not provided by intentional design, instead everything must be wrapped in a class. [31]

An easy solution to this problem was devised, namely linking the simulator to every simulation participant (object, event, activity etc). This method offers both benefits and drawbacks. On the one hand, it allows you to isolate the simulation with all the necessary objects in the system for each scenario. On the other hand, it creates high interconnection within the system. The simulator is passed as an argument to the constructor for each simulation class (simulation participant).

3.4.2. Function objects

Since JS is a dynamic type language, whereas Java is a statically typed language, there are certain challenges during the mapping of particular classes. Let's look at an example:

```
if (typeof AT.duration === "function") {
     acty.duration = AT.duration();
} else {
   acty.duration = AT.duration;
}
```

Listing 3.2: Example of the dynamic typing in JavaScript from OESjs framework

As you can see in the example above, the attribute ”duration” of the ”AT” object (which is activity type) could be either a functional expression or a numeric value. The reproduction of the given code in Java is almost impossible, because the JVM-compiler must surely know if he is working with an object or a function (method) of a certain object. Java release version 1.8 has introduced functional programming support (with ”functional interfaces”), which allows us to invent a workaround for a given situation.

When it comes to functional interfaces, there are a few that are particularly important: Consumer, Supplier, and Function. As it can already be inferred from the name, the Consumer interface ”consumes” the input supplied to it and returns no output. A similar assumption works for the Supplier interface - it ”supplies” some result without the need to input parameters. The Function interface can be considered as a combination of the Consumer and Supplier, so it takes one argument and produces a result out of it. Now we can create an object implementing one of the given interfaces, which essentially work as functions, and assign it to a variable. So it turns the code from 3.2 into:

```
if (acty.getDurationFunc() != null) {
  acty.setDuration(AT.getDurationFunc().get());
 } else {
  acty.setDuration(AT.getDuration());
 }
```

Listing 3.3: Example of functional programming in OES Java framework

Class AT (activity type) has now two attributes - ”durationFunc” implementing Supplier interface and ”duration” of the type Number. A Supplier has a generic get() method, which returns a value of predefined type.

```
public abstract class aCTIVITY extends eVENT {
   private Number duration;
   protected Supplier<Number> durationFunc;
   ...
}
```

Listing 3.4: Activity class with extended attributes in framework

```
public class MakePizza extends aCTIVITY {
   public MakePizza(Simulator sim, Number id, Number startTime) {
      this.durationFunc = () -> Rand.triangular(3, 6, 4);
...
}
```

Listing 3.5: Example of implementation of activity class with duration function in Make-and-Deliver Pizza example model

3.4.3. Constructor property and activity types

The constructor property in JavaScript returns a reference to the Object `constructor` function that created the instance object. Mostly this property is used for defining a function as a function-constructor with further calling it with new and prototype-inherits chain. [27] The (introspective/reflective) constructor property also allows to retrieve the class of a typed/constructed object.

In OESjs, the constructor property is used to access standard constructor parameters, manipulate them and instantiate new objects of a certain (activity) type. The following example illustrates how constructor properties are helpful for accessing parameters of a class and instantiating new objects in OESjs:

```
 AT = acty.constructor, // the activity's type/class
 ...
 // a string or a function returning a string
 if (AT.successorActivity) {
        const SuccAT = typeof AT.successorActivity === "function" ?
            sim.Classes[AT.successorActivity()] :
            sim.Classes[AT.successorActivity];
      succActy = new SuccAT();
      succActyResRoleNames = Object.keys( SuccAT.resourceRoles);
      actyResRoleNames = Object.keys( AT.resourceRoles);
      ...
}
```

Listing 3.6: Use of constructor property in OESjs

As you see, after setting `AT = acty.constructor`, AT represents the activity class of `acty` and the expression `AT.successorActivity` allows to test if this class has a static property `successorActivity`.

Unfortunately, in Java there is no way to derive an object's constructor and assign it to a variable, which could be used later for the instantiation of a new object (of the same type).

By design, the simulator holds a map of all the object, event, and activity classes of a particular model, allowing the user to retrieve these classes by their names. This map has activity names as keys and activity classes as values. In 3.6 you may see that, given an activity class, one can instantiate new objects and access static properties. In the JS snippet, it looks easy and simple to implement, but the same implementation in Java is complicated for a number of reasons:

- Having the map specification, it could be assumed that the corresponding map in the Java implementation would be `Map, Class extends aCTIVITY>>`, but the Class class [32] contains only some information about the class it refers to, so its use for instantiation and accessing of concrete static values is very uncomfortable and subjected to errors (e.g. NoSuchMethodException).

- It is possible to get a class by name in Java, primarily using `forName` method [33], but the compiler can't guarantee what kind of class you are going to get. If we

know what kind of class we are going to get, we will have to cast it (even still, we get unchecked cast warnings because the cast is not safe). The object type casting couldn't be predefined in our code, since we will retrieve different types (classes) of objects contained in a particular simulation.

- All instances of the class "aCTIVITY" share a set of static properties, but only at the level of a particular (child) class (for example, the "resource role" property). This suggests that since we are aware that each instance of the "aCTIVITY" class has such static properties, we are not permitted to declare them at the level of the parent "aCTIVITY" class; otherwise, these properties would be shared by all members of child classes, reducing the quality of differentiation. What we can do is declare abstract getter-methods that ensure that each child class exposes such static attributes at the class level and that they may be accessed statically.

- There is a way to create an instance of a particular class given the class name (dynamic) and passing parameters to its constructor using `newInstance()` method of the `Class<T>` generic class. However, we can not be certain in this manner, that objects will be instantiated in the right (predefined) way for each and every activity type, thereby provoking a lot of exceptions at run time. [29]

To address the issues stated and recreate the architecture of OESjs, the following decisions were made:

1. Defined abstract methods of aCTIVITY class, which are in common for all activity types:

```
public abstract class aCTIVITY extends eVENT {
   public abstract tASKqUEUE getTasks();
   public abstract void setTasks(tASKqUEUE t);
   public abstract Map<String, List<rESOURCE>> getResources();
   public abstract void setResources(Map<String, List<rESOURCE>> res);
   public abstract Map<String, rESOURCErOLE> getResourceRoles();
   public abstract void setResourceRoles(Map<String, rESOURCErOLE>
       resRoles);
   public abstract List<rESOURCE> get(String resRoleName);
   public abstract void put(List<rESOURCE> rESOURCEs, String resRoleName);
   public abstract void delete(String resRoleName);
   public abstract String getSuccessorActivity();
   public abstract aCTIVITY newInstance();
   ...
}
```

Listing 3.7: Abstract methods of activity class

This ensures that all activity types will implement these methods (which are mostly getters and setters), thus having specific attributes, like resources or resource roles, defined per class.

2. Added a public method for each activity class to dynamically instantiate new objects of this class:

```
public class MakePizza extends aCTIVITY {
...
   public aCTIVITY newInstance() {
      return new MakePizza(this.getSim());
   }
...
}
```

Listing 3.8: The instantiation of activity objects (MakePizza)

A single object of an activity type may now be able to create new objects of the same type.

3. Filled in the map of activity types with mocked objects for each simulation. This allows us to take advantage of the functionality provided by the two previously mentioned technical decisions:

```
Simulator sim = new Simulator();
sim.getAClasses().put("GoToLoadingSite", new
    GoToLoadingSite(sim,0,0,null));
sim.getAClasses().put("Load", new Load(sim,0,0,null));
sim.getAClasses().put("Haul", new Haul(sim,0,0,null));
sim.getAClasses().put("GoBackToLoadingSite", new
    GoBackToLoadingSite(sim,0,0,null));
sim.getAClasses().put("GoHome", new GoHome(sim,0,0,null));
sim.getAClasses().put("Dump", new Dump(sim,0,0,null));
```

Listing 3.9: Filling the map of activity types with mocked objects (Load-Haul-Dump model)

3.4.4. HTML access

JavaScript provides the possibility to manipulate HTML elements, e.g. by accessing a node in the Document Object Model (DOM) using an ID or class. The same approach can be obtained in the OESjs application during representation of simulation results in UI (see `simulatorUI.js` file in [3]).

Unfortunately, Java does not have any backward compatibility with HTML files by default. However, this challenge was handled by using Thymeleaf (see section 3.2.4) as an external library for processing and representation of HTML templates.

Thymeleaf, together with Spring MVC, allows us to manage web page templates for the UI of OESjava application and fill these pages with different content in different ways. First, we need to place an HTML template in the special folder named ”templates” under the resource folder of the OES Java project. The next step will be the set up of the REST-controller, which will deliver a specified page to the client (browser). After a request

is processed at the endpoint, the Thymeleaf engine will take the specified HTML page, parse it, format the Thymeleaf attributes (if any are given), and deliver the completed page to the client.

The example of the HTML template is provided in the appendix (section A.3). At the REST-endpoint, we will assign values to the placeholders inside the HTML template, annotated with curly brackets and dollar-sign in front of them.

```
@PostMapping("/core2/loadhauldump")
public String runLoadHaulDump(SimulationSettingsDTO dto, Model model) {
   model.addAttribute("type", dto.getType());
   model.addAttribute("hasLogs", dto.isSimulationLog());
   loadHaulDumpSimulation.run(dto, model);
   return "loadhauldump";
}
```

Listing 3.10: Template processing inside endpoint for the Load-Haul-Dump model

For this purpose, an implicit model object is added to the method's argument list. This model object is handled by Spring Boot and will be provided to the view automatically. Any type of data object can be included in the model (in our case, a custom DTO-object is used). Also, some of the placeholders will be assigned during the execution of the simulation.

```
public void run(SimulationSettingsDTO dto, org.springframework.ui.Model m) {
   ...
   m.addAttribute("stat", sim.getStat().getSimpleStat());
   ...
   m.addAttribute("actStat", sim.getStat().getActTypes());
   if(dto.isSimulationLog()) {
      m.addAttribute("logs", SimulatorUI.getLogs());
   }
   ...
}
```

Listing 3.11: Template processing inside simulation component

Thymeleaf also has the ability to deal with the Java collections framework, which is a huge advantage of the build-in template engine. It means that all standard methods of Java collection interfaces can be invoked inside the template, like viewing keys contained in a map object or getting the number of elements in a collection. This feature will be used e.g. for retrieving of simulation statistics.

3.4.5. Simulation Server

JavaScript is a single-threaded environment, which means multiple scripts can not run at the same time. A user has just one thread per browser tab (with some exceptions) to do the work of rendering sites and running JavaScript. Since the browser's main thread doesn't allow longer computations and may become incredibly overworked, the JS Web

Worker was integrated into the OESjs. It is in charge of executing the simulation loop.

So JavaScript is a client side language, which is specially designed to be run on the client browser from its originality, but Java is not designed for this purpose. To enable adequate communication between a client and server, the decision to build a Web service following the Representational state transfer (REST) software architectural style was made. Moreover, it must be an intermediary Application Programming Interface (API), that defines the endpoint and methods allowed to access/submit data to the server.

First of all, the homepage and corresponding endpoint for it were defined:

```
@Controller
public class ObjectEventSimulationEndpoint {
   @RequestMapping("/")
   public String index() {
       return "index";
   }
   ...
}
```

Listing 3.12: Endpoint for homepage

It will deliver the "index.html" file from the available templates in the Thymeleaf resource folder (see section A.4 in the appendix). Users may access the settings-page of different simulation scenarios from the homepage. The setting-page, together with a DTO object to be constructed, will be delivered from the server through an HTTP GET-request to a particular endpoint:

```
@Controller
public class ObjectEventSimulationEndpoint {
   @GetMapping("/core2/makeanddeliver")
   public String setupMakeAndDeliver(Model m) {
      m.addAttribute("scenario", new SimulationSettingsDTO());
       return "makeanddeliver-settings";
   }
   ...
}
```

Listing 3.13: Endpoint returns the setting page for Make-and-Deliver simulation and injects a DTO object inside Thymeleaf-template

Afterwards, the user configures the simulation parameters, which includes the setup of SimulationSettingsDTO. By pressing the "submit" button, the client sends a HTTP POST-request with SimulationSettingsDTO in the request body. POST-requests are handled in the same way as GET-requests by the server. The server performs a simulation based on the parameters set by the user.

```
@PostMapping("/core2/makeanddeliver")
@Controller
public class ObjectEventSimulationEndpoint {
    public String runMakeAndDeliver(SimulationSettingsDTO dto, Model model)  {
```

```
        model.addAttribute("type", dto.getType());
        model.addAttribute("hasLogs", dto.isSimulationLog());
        makeAndDeliver.run(dto, model);
          return "makeanddeliver";
    }
    ...
}
```

Listing 3.14: Endpoint evaluates response and runs the simulation loop using the parameters provided for Make-and-Deliver Pizza simulation model

In the same way, the initialization of each scenario is configured, what allowes the user to specify the necessary parameters and deliver the results of the simulation run. The same UI is used in OESjs implementation.[30].

3.5. Validation of the reconstruction result

3.5.1. Reconstruction approach

The reconstruction of OESjs framework in Java language was done in an iterative manner:

1. Implement the OES Core 0, which corresponds to the minimal architecture for an OE simulator
2. Extend the application with OES Core 1 features, such as a seedable random number generator, model parameters, multiple scenarios and/or experiment types per model, etc
3. Extending the OES Core 1 simulator by adding support for activities, which are composite events with some duration.

Between each of these iterations, the confirmation must be ensured by examination and provision of objective evidence that the reconstruction result conforms to specification, and that the particular requirements of each ”Core”-level can be consistently fulfilled.

Both the OESjs and OESjava simulators can generate a simulation log, which allows you to inspect the evolving states of a simulation run. Examining the simulation log can assist in understanding the dynamics of a model or in identifying logical errors.

3.5.2. Use of logs

In Java implementation, the contents of the simulation log can be controlled by overriding the ”toString” method for those objects that are to be displayed in the log. For instance, in the case of the Inventory Management model (Core 1) [34], the SingleProductShop object has following implementation of the ”toString” method:

```
@Override
public String toString() {
   return this.getClass().getSimpleName() + "{stock : " +
       this.getQuantityInStock() + "}";
}
```

Listing 3.15: Implementation of the "toString" method for SingleProductShop object

This results in the following simulation log:

Step	Time	System State	Future Events
0	0.0	{1=SingleProductShop{stock : 80}}	DailyDemand{quant : 25} @1
1	1.0	{1=SingleProductShop{stock : 55}}	DailyDemand{quant : 29} @2.0
2	2.0	{1=SingleProductShop{stock : 26}}	DailyDemand{quant : 27} @3.0 DailyDemand{quant : 10} @4.0
3	3.0	{1=SingleProductShop{stock : 0}}	DailyDemand{quant : 10} @4.0 Delivery{quant : 74} @5.0
4	4.0	{1=SingleProductShop{stock : 0}}	Delivery{quant : 74} @5.0 DailyDemand{quant : 9} @5.0
5	5.0	{1=SingleProductShop{stock : 74}}	DailyDemand{quant : 9} @5.0
6	5.0	{1=SingleProductShop{stock : 65}}	DailyDemand{quant : 26} @6.0

3.5.3. Validation methods

The simulation results were validated by comparing the actual log output of the Java implementation to the OESjs log output. In doing so, two methods of validation have been taken into account:

- **Standalone simulation**. Running a standalone simulation allows us to inspect the overall state of the system at each simulation step. Comparing results at such a level offers a sense of whether the application's logic is well implemented, all objects are well-defined, state changes occur according to specified rules, and so on.
- **Experiment with N replications**. In the simplest case, a simulation scenario is run repeatedly to be able to compute aggregate statistics. On top of a model, an experiment specifies the number of replications and a list of seed values (one for each replication). The resulting statistics are composed of the statistics for each replication complemented with summary statistics listing averages, standard deviations, min/max values and confidence intervals, as shown in the following table in Appendix A.5. Resulting statistics are a helpful signal for developers to assess if simulation works correctly under varied input settings and if outgoing results are similar on average to the source implementation, just as they are useful for experts in simulation for scenario analysis and decision-making.

3.5.4. (Semi-)Automatic verification of results

When a testcase is conducted with the assistance of a tool without requiring significant manual intervention, this is referred to as *automated testing* [26]. Automation tests enable the avoidance of recurrent manual efforts, which could be slower and prone to error. Numerous test scenarios can be automated and executed several times within

time and resource constraints, resulting in increased test coverage and output quality. To create automated test cases the Unit Test framework (see section 3.2.6) was used. To ensure that the code behaves logically as intended, a testcase is constructed with a certain checkpoint or verification condition. When the testcase is executed, the criteria/condition is either met or violated. The JUnit framework will provide a summary of the passed and failed test cases.

From requirement and analysis to design and development and maintenance, each phase should be accompanied by an appropriate testing phase. Code testing is an important step in order to ensure the durability of the program and the usage of optimized code.

Since each simulation model code has fixed (such as the one defined by input parameters) and randomized data (such as output statistic for each scenario run), two techniques for semi-automatic verification of reconstruction results could be used:

1. Add a check for correct model initialization - this can be a check for consistency of the number of steps (amount of time) spent on the execution of the scenario to the initial parameters; the number/characteristics of the objects involved in the scenario; the number of function calls; etc.

2. Explore the upper and lower bounds of the output values of statistics that appear as a result of the execution of the source code (of the OESjs framework). Based on this knowledge, define assets which will ensure whether the execution of the target code (OESjava) delivers a valid result - the one that is within a particular interval.

4. AnyLogic simulation models

4.1. Introduction

4.1.1. AnyLogic Enterprise Library

The AnyLogic development environment is based on the object-oriented programming paradigm. An active object in AnyLogic is an object with its own functioning and interaction with the environment. It can include an unlimited number of other active objects. The distinctive feature of AnyLogic is that this development environment is not limited to only one modeling paradigm. It provides different levels of abstraction, different styles and concepts, build models within different paradigms and mix them up when creating the new model, use previously developed modules compiled into libraries, also add and build own module libraries. Thanks to the built-in animation and visualization capabilities, models are more understandable and are representing the essence of the processes occurring in a simulated system and simplify model debugging. The AnyLogic simulation environment supports model design, development, documentation, computer experiments with the model, including various types of analysis - from sensitivity analysis to optimization of model parameters relative to a certain criterion. [36]

AnyLogic Enterprise Library provides a high-level interface for quickly creating discrete-event models using flowcharts. Graphical representation of systems by means of flowcharts is widely used in different research and production fields: manufacturing, logistics, service systems, business processes, computer and telecommunication network modeling, etc. AnyLogic allows to model with visual, flexible, extensible, reusable objects, both standard and developed. The Enterprise Library contains traditional objects: queues, delays, pipelines, resources, etc., so that the model is quickly built drag-and-drop and very flexibly parameterizable.

The implementation of standard Enterprise Library objects is open to the user, and their functionality can be expanded as desired, up to creating own libraries. To better understand how the library objects work and how to create new objects with the required functionality, it is necessary to inspect the code of the library objects.

4.1.2. Basic components

Let us begin by describing the fundamental components of AnyLogic that we will require for reconstructing the OES models. All information provided is accurate and consistent with the official documentation [37].

Source. Creates entities. Typically used to initiate the entity stream. Entity can be either a child of the base class ”Entity” or any user class that inherits from it. You may

configure the object to produce entities of different types by supplying the constructor of the appropriate class in the New Request argument, as well as the action to be done before the new entity exits the object and associating the new entity with a certain animation figure.

Figure 4.1.: Source component.

Sink. Disposes of all incoming entities. Typically used at the end of an entity's flow. To remove and destroy entities from the model, connect the output port of the last block of the process diagram to the Sink or Exit port.

Figure 4.2.: Sink component.

Delay. Agents are delayed for a certain amount of time. The delay duration is dynamically computed and may be random based on the current agent or other factors.

Figure 4.3.: Delay component.

Select output. According to the fulfillment of a particular (deterministic or probabilistic) condition, the object sends incoming agents to one of the two output ports. The condition may be determined by the agent or by some external variables. At the same moment, the incoming agent exits the SelectOutput object.

Figure 4.4.: Select output component.

Service. Seizes a given number of resource units, delays the agent, and releases the seized units. Is equivalent to a sequence Seize, Delay, Release and should be used if the agent does not need to do anything but execute a delay between seize and release. Most of parameters and extension points of these blocks are exposed by Service.

Figure 4.5.: Service component.

Resource pool. Defines a set of resource units that can be seized and released by agents using Seize, Release, Assembler, and Service flowchart blocks. The resources can be of three types:

- Static resources are bound to a particular location (i.e. node) within the network and cannot move or be moved. An example of a static resource would be an X-Ray room or a weigh bridge.
- Moving resources can move on their own, they can represent staff, vehicles, etc.
- Portable resources can be moved by agents or by moving resources. A portable U-Sound device or a wheelchair would be an example of a portable resource.

Moving and portable resources have a home place to which they can return or be returned at their discretion. Individual resource units in a pool can be animated, gather unit-based data, etc. You may create resource types for workers, equipment, etc. The agent can select a resource unit by assessing its characteristics.

Any resource unit can be busy or idle. Each resource pool block gathers utilization statistics, or the percent of busy units. Individual resource units always collect their individual utilisation information. You may also gather comprehensive information on each resource unit downtime, such as maintenance, breakdowns, and custom downtime.

Moving resources do not recognize each other as obstacles and will move over each other, which does not correspond to reality.

Figure 4.6.: Resource pool component.

4.2. Reconstruction of the OES models in AnyLogic

AnyLogic and other well-known DES tools do not support pure activity-based DES modeling or the Activity Network paradigm, but rather the Processing Network (PN) paradigm, in which processing items enter a system via arrival events and then ”flow through the system” to the end. However, the PN paradigm only encompasses a small but significant subset of discrete process models.

To create an activity-based DES model (or an Activity Network model) using AnyLogic, we must impose the PN paradigm on it and utilize the PN modeling notions of entrance nodes (referred to in AnyLogic as ”Sources”), processing nodes (referred to as ”Services”), and exit nodes (called ”Sinks”). A processing node is a sophisticated modeling element that comprises a processing station object (complete with an input buffer for arriving processing objects), a processing activity, and specific resource assignments.

4.2.1. Make and Deliver Pizza

Along with the simulation model represented by DPMN diagram in the Figure 2.7, we also require the data associated with a simulated scenario, such as the initial states of variables and objects, including resource pools, in order to execute the simulation. We begin with a standard setup that includes two order takers, six pizza makers, three ovens, and ten scooters.

As no entities flow through the real pizza service delivery system, we must assume a fictional abstract entity, such as ”the order,” that arrives at the order taker and subsequently takes the form of the requested pizza being delivered to the client.

A DPMN process design model 2.7 may be built using AnyLogic’s Process Modeling Library in the following steps:

1. A circle of exogenous events, such as OrderCall, is transformed into an entry node OrderCall (an AnyLogic ”Source” element). Due to the fact that the OrderCall event’s recurrence is specified case-by-case for each of the pizza service’s five hours of operation, we set the field Arrivals defined by to ”Rate schedule” and the field Rate schedule to the schedule that specifies the arrivals (”ArrivalSchedule”).

2. The following steps are used to implement the TakeOrder activity rectangle as a matching processing node (an AnyLogic ”Service” element) and :

a) Transformation of the performer role orderTaker:OrderTaker (abbreviated in the diagram as :OrderTaker) into a resource pool by creating an AnyLogic ”Resource Pool” with the name orderTakers and a capacity field equal to the size of the resource pool indicated by the simulation scenario selected (like 2 in the baseline scenario) and setting the Seize field to ”units from the same pool” and the Resource pool field to the previously defined pool orderTakers (by selecting it from the drop-down list), while maintaining the default value ”1” in the field number of units, which corresponds to the performer role :OrderTaker's default resource cardinality constraint of ”exactly one”.

b) Due to the unbounded nature of the TakeOrder task queue, the yes/no field ”Maximum queue capacity” must be checked.

c) The field ”Delay time” is set to the duration of the activity in an AnyLogic Java expression: uniform (1,4).

d) Because the TakeOrder activity includes a waiting timeout, the yes/no field ”Queue: exit on timeout” must be selected in the Advanced section of the ”Service” element, and the field Timeout is set to an AnyLogic Java expression matching to the activity's waiting timeout value: uniform discr(3,6).

e) Finally, the LostOrder event circle is implemented as a LostOrder exit node (an AnyLogic ”Sink” element) connected to the timeout out-port of the ”Service” element.

3. The MakePizza activity rectangle is turned into a matching processing node (an AnyLogic ”Service” element) by doing the following:

a) The performer role pizzaMakers:PizzaMaker is turned into a resource pool assignment by defining an AnyLogic ”Resource Pool” with name pizzaMakers and set its field Capacity to the resource pool size defined by the chosen simulation scenario (like 6 in the baseline scenario) and Set the Seize field to ”(alternative) resource sets” and then create an entry in the field Resource sets (alternatives) by selecting the pool pizzaMakers from the drop-down list and changing its default value ”1” for the resource cardinality to ”2” (because ”exactly 2” pizza makers are required).

b) The resource role oven:Oven is converted into a resource pool assignment by creating an AnyLogic ”Resource Pool” with the name ovens and setting its field Capacity to the resource pool size defined by the selected simulation scenario (such as 3 in the baseline scenario) and creating another entry in the field Resource sets (alternatives) by selecting the pool ovens from the dropdown list (and leaving its default value ”1” for the resource cardinality).

c) Because the MakePizza task queue is infinite, the yes/no parameter Maximum queue capacity must be selected.

d) The field Delay time is set to an AnyLogic Java expression equal to the duration value of the activity: triangular (3, 6, 4).

4. The DeliverPizza activity rectangle is turned into a matching processing node (an AnyLogic ”Service” element) by doing the following:
   a) To simplify matters, the performer role scooterDriver:ScooterDriver (abbreviated in the diagram as :ScooterDriver) has been combined with the resource role scooter:Scooter (abbreviated by :Scooter).
   b) The resource role scooter:Scooter is converted to a resource pool assignment by creating an AnyLogic ”Resource Pool” named scooters, setting its Capacity field to the size of the resource pool defined by the selected simulation scenario (for example, 10 in the baseline scenario), setting the Seize field to ”units of the same pool”, and setting the Resource pool field to the previously defined pool scooters (by selecting it from drop-down list)
   c) Due to the unbounded nature of the DeliverPizza task queue, the yes/no parameter Maximum queue capacity must be checked.
   d) The field Delay time is set to an AnyLogic Java expression corresponding to the activity’s duration value: triangular(10, 30, 15).

The following AnyLogic process diagram illustrates the outcome of these model implementation steps:

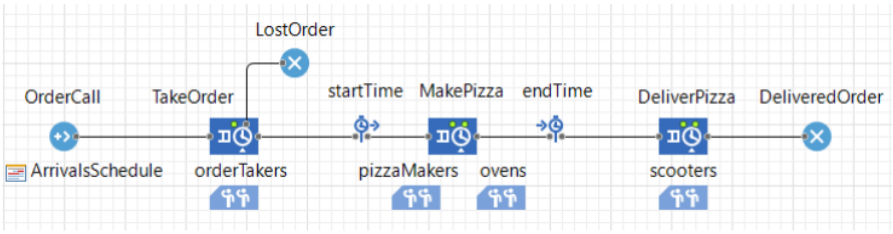


Figure 4.7.: An AnyLogic process diagram for the Make-and-Deliver-Pizza business process.

Notably, the related processing node is included in a TimeMeasureStart-TimeMeasureEnd pair of time measurement nodes for monitoring the cycle time of MakePizza operations (startTime and endTime).

4.2.2. The Load-Haul-Dump Model

As previously stated, AnyLogic does not support Activity-Based Discrete Event Simulation, but only Processing Network (PN) simulations with ”entities flowing through the system.” Therefore, an option is to use the five trucks of the haul service company as the entities (referred to as ”agents” in AnyLogic), which arrive at a truck entry node and ”flow” through the processing nodes Load, Haul, and Dump before they exit the system. While this technique is rather unnatural in that it imposes the PN paradigm on the presented issue, it enables us to use AnyLogic.

We can only consider the following elements based on the information design model (Figure 2.11):

1. All variables in the model are implemented as AnyLogic variables (defined in Main).

2. The duration functions provided in activity types are implemented as the Delay times of anyLogic Delay or Service components.

3. The resource roles assigned to activity types are implemented using Seize elements or Seize information items included inside Service elements (if they had not to be transformed into entities, like the trucks of the Load-Haul-Dump model).

For example, the activity type Load is implemented as a Service element together with the resource roles wheelLoader and truck (same as for to the Make-and-Deliver model, see previous chapter 4.2.1):

1. Its duration function, which samples the U(10,20) distribution, is specified as ”uniform(10,20)” in the Load Service element’s Delay time field.

2. its resource role wheelLoader entered in the form of a resource pool wheelLoaders in the Resource pool field of the Load Service element, and its resource cardinality constraint of exactly one entered as ”1” in the field number of units.

While the DPMN process starts with the event HaulRequest, the AnyLogic process starts with the entry node truckEntry (a ”source” element) with the field ”Arrivals defined by” set to ”Calls of inject() function,” which indicates that the inject function must be invoked in the Main ”On startup” function in the following manner: truckEntry.inject(5).
To cover all bases, the DPMN model’s GoToLoadingSite activity is omitted.

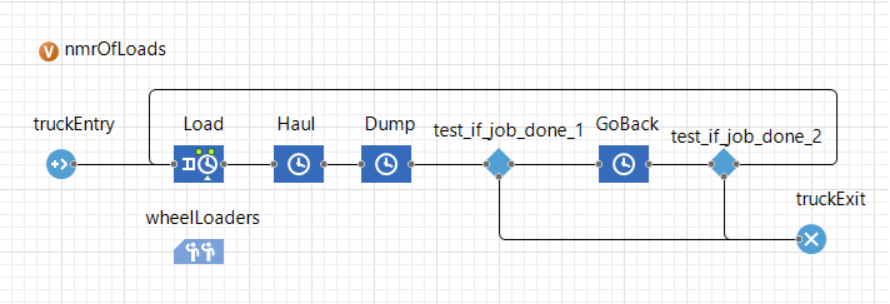


Figure 4.8.: An AnyLogic process diagram for the Load-Haul-Dump business process.

As explained above, the activity type Load is implemented as a Service element with a ”Delay time” of ”uniform(10,20)” and the field ”Resource pool” set to ”wheelLoaders” with ”number of units” set to 1. All other activity types are implemented as Delay elements (with a ”Delay time” provided by their duration function defined in the class model).
The two decision diamonds of the DPMN model are implemented with corresponding ”SelectOutput” elements called ”test_if_job_done_1” and ”test_if_job_done_2”.

5. Mapping of OES models to AnyLogic models

5.1. Introduction

A model transformation is the process of converting a model into another form according to a set of transformation rules. According to the output type or target language of the process, the model transformation can be classified as a model to model (M2M) transformation, a model to text (M2T) transformation, or a model to code transformation (M2C or code generation) [42]. An M2M transformation translates models from an input modeling language to an output modeling language, possibly on a different level of abstraction.

Research has produced various means to develop solutions for representing the different concerns of modeling languages. Lately, two different language implementation techniques have been distinguished: internal and external [41]. While the first one makes use of the host language's features and is constrained in terms of expressiveness by the host programming language, the second one has its own stand-alone syntax that necessitates the use of tools to convert its models into machine-processable representations, as result, there is more flexibility in language definition, but greater afford during the development and providing of necessary language-related tooling. If we consider the scope of this thesis, we can classify AnyLogic as an external language, because it has its own syntax for creating models (diagrams), whereas the OESjs/OESjava could be considered as internal modeling languages, because they both have a fluent API in host programming language (Java/JavaScript) and do not require any external tool for simulation result interpretation.

Many different modeling paradigms and techniques are supported by the AnyLogic software, such as event scheduling [39], network of processing nodes [38], and agent-based simulations [40], among others. At the same time, the Object Event Simulation paradigm encompasses the Event-Based Simulation paradigm, which includes Event Scheduling and Future Event List, as well as Processing Network Simulation, which includes activities, and Agent-Based Simulation, which is a high-level version of Discrete Event Simulation. Thus, AnyLogic and OES are intended to have some similar ontological concepts of discrete-event simulation modeling methodologies.

As shown in Chapter 4, it is possible to reconstruct and execute given models in AnyLogic to realize their dynamic semantics. Now, in order to determine mapping rules between the two modeling languages, we will attempt to abstract away from the essence of each modeling language and make use of the common ideas and elements shared between them.

5.2. Mapping of model diagrams

For model diagrams, a combination of an OE class model and a DPMN process model could be used to demonstrate how each element of the source model is mapped to one or more elements (or patterns) in the target model.

5.2.1. Make and Deliver Pizza model

An OE class model and a DPMN process model for the Make-and-Deliver Pizza (M&DP) case study were presented in Chapter 2.2.1. Later in Chapter 4.2.1, the model was reconstructed using the AnyLogic software. Given these results, we can compare the modeling objects in the source and target models. Concerning M&DP model, the following observations were made:

- An exogenous event circle OrderCall is turned into an AnyLogic "Source" element. The event recurrence is defined by the "ArrivalSchedule" element, which is assigned to the "Rate schedule" field of the "Source" element.
- The performer and resource roles OrderTaker, ScooterDriver, Oven and PizzaMaker were turned into an AnyLogic "Resource Pool" with the corresponding name.
- The TakeOrder, MakePizza and DeliverPizza resource-constrained activities were turned into an AnyLogic "Service" element with the connected corresponding resource pool(s). E.g. the MakePizza Service is connected to the "pizzaMakers" and "ovens" resource pools through the "resource sets" attribute, which is represented by a drop-down list.
- The LostOrder event circle (final activity) is turned into an AnyLogic "Sink" element attached to the "Service" element's timeout out-port.
- An AnyLogic "Sink" element ("Delivered order") was appended to the DeliverPizza activity to indicate that it is the network's final activity.

Besides that, there are a number of non-graphic elements/attributes derivable from the DPMN/OEM diagram, which have been transferred to the target AnyLogic model:

- The duration of an activity is turned into an AnyLogic Java expression ("Delay time" field). AnyLogic offers a set of probability distribution functions that will return random values distributed according to various laws each time they are called on a "Delay time" field [48].
- The lost orders (due to long waiting times) are modeled using an AnyLogic Java expression corresponding to the waiting timeout value for the activity.
- The constraint on resource pool cardinality is specified in the field "number of units" of the resource sets property of the associated Service element.
- The value of a Resource pool's Capacity attribute must be equal to the size of the resource pool described by the selected simulation scenario.

5.2.2. The Load-Haul-Dump model

Chapter 2.2.2 introduced an OE class model and a DPMN process model for the Load-Haul-Dump scenario, which was also recreated using the AnyLogic software in Chapter 4.2.2. Now we can verify the validity of the preceding assumptions in Chapter 5.2.1. by comparing the source and target models. The following observations were made regarding the Load-Haul-Dump model:

- Resource role WheelLoader was turned into an AnyLogic "Resource Pool" element with a corresponding name, which conforms to the transformation made in the M&DP model, but the Trucks resource role was turned into an entity which is "flowing" through the processing network and doesn't require any resource pool for it. Whereby in M&DP model a new "abstract" entity with the name OrderCall was introduced.
- The resource-constrained activity "Load" was transformed into an AnyLogic "Service" element, along with the associated "wheelLoaders" resource pool. What is different from the M&DP model is the transformation of Haul and Dump resource-constrained activities into Delay elements (with corresponding duration function) of the AnyLogic model, since there exists no resource pool for trucks, which could be associated with a Service element.
- The two decision diamonds of the DMPN model were turned into corresponding "SelectOutput" elements of AnyLogic.
- While the DPMN process begins with the event HaulRequest, the AnyLogic process begins with the "Source" element named "truckEntry", with the field "Arrivals defined by" set to "Calls of inject() function," which indicates that the inject function must be invoked in the Main "On startup" function in the following manner: `truckEntry.inject(5)`.
- The Load-Haul-Dump network may be considered circular, as there is no end activity in the sequence, but the AnyLogic processing network paradigm requires an exit node ("Sink" element) for processing entities to escape the system, and therefore the "Sink" element with the name "truckExit" was constructed.

All other non-visible attributes of the AnyLogic model were implemented identically to the M&DP model.

This example demonstrates how converting one of the resource roles to an entity (referred to as a "agent" in AnyLogic) requires a fundamentally different model structure, imposing the Processing Network paradigm on the given problem.

5.3. Mapping of model code

5.3.1. The ALP-file format of AnyLogic

Because AnyLogic is an external modeling language, it has its own syntax and file format for model creation. Anylogic saves the models as standard XML files with the .ALP

ending, which makes it easy to make manual or algorithmic changes to the model in concern. Blocks and agents, connections between blocks, statistics monitors, input readers, output writers are all stored in the Anylogic XML simulation model file, which also has a tree-like structure, where data is stored as elements (nodes). An element's attributes describe the element's type and its characteristics. The attributes may contain several lines of code defining the block's behavior in various circumstances and states.

No specification for the ALP format was discovered during online and literature searches; also, it is not intended to be a 'API' for model developers to interact with (and the developers of AnyLogic routinely make non-backwards-compatible changes to the format between releases). So in this section, only a presumptive specification of the ALP file structure (for version 8.5.7) will be presented, with no validity claim on it.

Moreover, only the elements important for further research will be presented in this section. The ALP file contains a lot of information that is not important in the context of activity-based simulation, such as the position of objects in space, length units, colors, etc. Also, in the ALP file there is plenty of technical information (such as special object parameters and IDs, package names, and so on), which is neither important for the reconstruction of the simulation nor explainable without open-source documentation.

The most general and introductory element of an ALP file is `EnvironmentProperties`. The name is self-descriptive and this element contains a couple of important parameters, such as `SpaceType` (one of CONTINUOUS, DISCRETE, GIS, UNDEFINED) and `StepDurationCode` (enables discrete time steps with a given duration) [35].

```
<EnvironmentProperties>
   <EnableSteps>false</EnableSteps>
   <StepDurationCode Class="CodeUnitValue">
      <Code><![CDATA[1.0]]></Code>
      <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
   </StepDurationCode>
   <SpaceType>CONTINUOUS</SpaceType>
   ...
</EnvironmentProperties>
```

Listing 5.1: Example of EnvironmentProperties

The main building block of AnyLogic models is `EmbeddedObjects` element, which wraps `EmbeddedObject` elements. Each `EmbeddedObject` may have parameters, variables, ports, events, and types. The type of an EmbeddedObject is defined by `ActiveObjectClass` tag and references the package that contains a corresponding object class (type): Service, ResourcePool, Delay, etc. The name of an object is specified through `Name` attribute. An embedded object may have the `Parameters` block with nested parameters of its class. See the specification of the "Haul"-Service (from the Haul-Load-Dump model) in the source ALP file:

```
<EmbeddedObject>
   <Id>1538531166845</Id>
   <Name><![CDATA[Load]]></Name>
```

```
    <ActiveObjectClass>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
      <ClassName><![CDATA[Service]]></ClassName>
    </ActiveObjectClass>
    <Parameters>
      <Parameter>
        <Name><![CDATA[seizeFromOnePool]]></Name>
        <Value Class="CodeValue">
          <Code><![CDATA[true]]></Code>
        </Value>
      </Parameter>
      <Parameter>
        <Name><![CDATA[resourceSets]]></Name>
      </Parameter>
      <Parameter>
        <Name><![CDATA[resourcePool]]></Name>
        <Value Class="CodeValue">
          <Code><![CDATA[wheelLoaders]]></Code>
        </Value>
      </Parameter>
      <Parameter>
        <Name><![CDATA[queueCapacity]]></Name>
        <Value Class="CodeValue">
          <Code><![CDATA[1]]></Code>
        </Value>
      </Parameter>
      <Parameter>
        <Name><![CDATA[maximumCapacity]]></Name>
        <Value Class="CodeValue">
          <Code><![CDATA[true]]></Code>
        </Value>
      </Parameter>
      <Parameter>
        <Name><![CDATA[delayTime]]></Name>
        <Value Class="CodeUnitValue">
          <Code><![CDATA[uniform(10,20)]]></Code>
          <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
        </Value>
      </Parameter>
      <Parameter>
        <Name><![CDATA[onEnter]]></Name>
        <Value Class="CodeValue">
          <Code><![CDATA[nmrOfLoads--;]]></Code>
        </Value>
      </Parameter>
      ...
    </Parameters>
    ....
  </EmbeddedObject>
```

Listing 5.2: The definition of ”Haul”-Service in ALP file

```
<EmbeddedObject>
   <Name><![CDATA[wheelLoaders]]></Name>
   <ActiveObjectClass>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
      <ClassName><![CDATA[ResourcePool]]></ClassName>
   </ActiveObjectClass>
   <GenericParameterSubstitute>
      <GenericParameterSubstituteReference>
         <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
         <ClassName><![CDATA[ResourcePool]]></ClassName>
         <ItemName><![CDATA[1412336243135]]></ItemName>
      </GenericParameterSubstituteReference>
   </GenericParameterSubstitute>
   ...
</EmbeddedObject>
```

Listing 5.3: The definition of ”wheelLoaders” resource pool in ALP file

The `Connector` element is used to define how EmbeddedObjects are connected to one another. All Connectors are defined inside `Conntectors` tag. Since each connection has a source and a target (sometimes more than one), these attributes are declared inside the Connector element, referring to the object name and type using `SourceEmbeddedObjectReference`, `SourceConnectableItemReference` and `TargetEmbeddedObjectReference`, `TargetConnectableItemReference` tags respectively.

```
<Connectors>
   <Connector>
      ...
      <SourceEmbeddedObjectReference>
         <PackageName><![CDATA[ch3_earth_d1]]></PackageName>
         <ClassName><![CDATA[Main]]></ClassName>
         <ItemName><![CDATA[Haul]]></ItemName>
      </SourceEmbeddedObjectReference>
      <SourceConnectableItemReference>
         <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
         <ClassName><![CDATA[Delay]]></ClassName>
         <ItemName><![CDATA[in]]></ItemName>
      </SourceConnectableItemReference>
      <TargetEmbeddedObjectReference>
         <PackageName><![CDATA[ch3_earth_d1]]></PackageName>
         <ClassName><![CDATA[Main]]></ClassName>
         <ItemName><![CDATA[Load]]></ItemName>
      </TargetEmbeddedObjectReference>
      <TargetConnectableItemReference>
         <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
```

```
        <ClassName><![CDATA[Service]]></ClassName>
        <ItemName><![CDATA[out]]></ItemName>
      </TargetConnectableItemReference>
      ...
   </Connector>
   ...
</Connectors>
```

Listing 5.4: The connector between ”Load” and ”Haul” Delays in the Load-Haul-Dump model

The simulation variables (e.g. for statistic calculations) are defined under `Variables` block, see an example below.

```
<Variables>
   <Variable Class="PlainVariable">
      <Name><![CDATA[nmrOfLoads]]></Name>
      <Properties SaveInSnapshot="true" Constant="false" AccessType="public"
          StaticVariable="false">
         <Type><![CDATA[int]]></Type>
         <InitialValue Class="CodeValue">
            <Code><![CDATA[66]]></Code>
         </InitialValue>
      </Properties>
      ...
   </Variable>
   ...
</Variables>
```

Listing 5.5: The definition of ”nmrOfLoads” statistic variable in the Load-Haul-Dump model

`Schedule` element allows to track moments specified using a time table with some period. For example, the schedule of order arrival rates for M&DP model is defined as follows:

```
<Schedules>
   <Schedule>
      <Name><![CDATA[ArrivalsSchedule]]></Name>
      <ValueType>double_</ValueType>
      <UnitType><![CDATA[RATE]]></UnitType>
      <UnitOfValue Class="RateUnits"><![CDATA[PER_HOUR]]></UnitOfValue>
      <DefaultValue>0.0</DefaultValue>
      <RepresentationMode>intervals</RepresentationMode>
      <RepresentationType>timeUnits</RepresentationType>
      <SnapToTimeUnits>minutes</SnapToTimeUnits>
      <Intervals>
         <Interval>
            <Start>0</Start>
            <End>3600000</End>
```

```
            <Value>10.0</Value>
         </Interval>
         <Interval>
            <Start>3600000</Start>
            <End>7200000</End>
            <Value>90.0</Value>
         </Interval>
         <Interval>
            <Start>7200000</Start>
            <End>10800000</End>
            <Value>40.0</Value>
         </Interval>
         <Interval>
            <Start>10800000</Start>
            <End>14400000</End>
            <Value>10.0</Value>
         </Interval>
         <Interval>
            <Start>14400000</Start>
            <End>18000000</End>
            <Value>5.0</Value>
         </Interval>
      </Intervals>
      ...
   </Schedule>
</Schedules>
```

Listing 5.6: Abstract class ExogenousEvent and its implementation in the Make-and-Deliver Pizza model

5.3.2. Relevant Java model classes/attributes

Because OESjava provides a fluent API in the Java programming language, it is considered an internal modeling language. The components of the OESjava framework that may potentially be used to map/transform the model into an AnyLogic model will be discussed in this section.

Let's start with a general definition of model characteristics. Characteristics such as the underlying simulation time unit can be specified through an instance of the Model class connected to the running simulation. The time characteristics could be mapped into an EnvironmentProperties AnyLogic element, similarly to the code snippet 5.1.

```
Model model = new Model();
model.setName("Load-Haul-Dump");
model.setTime(Time.CONT);
model.setTimeUnit(TimeUnit.min);
model.setObjectTypes(List.of(Truck.class, WheelLoader.class));
model.setEventTypes(List.of(HaulRequest.class));
model.setActivityTypes(Set.of("GoToLoadingSite","Load","Haul","Dump",
```

```
"GoBackToLoadingSite","GoHome"));
...
Simulator sim = new Simulator();
sim.setModel(model);
```

Listing 5.7: Initialization of the Load-Haul-Dump simulation model

The Model class contains attributes that are valuable for model reconstruction - the potentially relevant object, event, and activity types. These attributes are good indicators of what elements must be present in the target model, otherwise it will be wrong or incomplete.

The objects extending the "ExogenousEvents" class are mostly modeled as a Source element in AnyLogic representation, since they are not caused by any causal regularity of the system under investigation and, therefore, have to be modeled with a recurrence function that allows one to compute the time of the next occurrence of an event of that type. Important attributes of this class for further reconstruction include: name of event, reference to the next event (successor activity), recurrence value.

```
public abstract class ExogenousEvent extends eVENT {
   public ExogenousEvent(Simulator sim, Number occTime, Number delay) {
      super(sim, occTime, delay, null, null);
   }
   public abstract Number getSuccessorActivity();
   public abstract Number reccurence();
   ...
}

public class ExampleEvent extends ExogenousEvent {
   public ExampleClass(Simulator sim, Number occTime, Number delay) {
      super(sim, occTime, delay);
   }
   @Override
   public String getSuccessorActivity() {
      return "ExampleSuccessorActivity";
   }
   @Override
   public Number reccurence() {
      return Rand.uniform(1,6);
   }
   ...
}
```

Listing 5.8: An example of an exogenous event class, derived from the abstract class ExogenousEvent (ExogenousEvent.java) in OESjava

A class inherited from the ExogenousEvent-class could be generalized to the following EmbeddedObject element in the ALP-file:

```
<EmbeddedObject>
```

```
   <Name><![CDATA[ExampleEvent]]></Name>
   <ActiveObjectClass>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
      <ClassName><![CDATA[Source]]></ClassName>
   </ActiveObjectClass>
   <GenericParameterSubstitute>
      <GenericParameterSubstituteReference>
         <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
         <ClassName><![CDATA[Source]]></ClassName>
      </GenericParameterSubstituteReference>
   </GenericParameterSubstitute>
   <Parameters>
      <Parameter>
            <Name><![CDATA[arrivalType]]></Name>
            <Value Class="CodeValue">
               <Code><![CDATA[self.INTERARRIVAL_TIME]]></Code>
            </Value>
         </Parameter>
         <Parameter>
            <Name><![CDATA[rate]]></Name>
            <Value Class="CodeUnitValue">
               <Code><![CDATA[100]]></Code>
               <Unit Class="RateUnits"><![CDATA[PER_HOUR]]></Unit>
            </Value>
         </Parameter>
         <Parameter>
            <Name><![CDATA[interarrivalTime]]></Name>
            <Value Class="CodeUnitValue">
               <Code><![CDATA[uniform(1, 6)]]></Code>
               <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
            </Value>
         </Parameter>
      ...
   </Parameters>
   ...
</EmbeddedObject>
```

Listing 5.9: Definition of exogenous event ExampleClass in ALP

Since OES follows the activity network paradigm, one of the main components of any model is activities, which are composite events with some duration. In the Java implementation, there is an abstract class aCTIVITY, which must be inherited by all activities involved in the model. Therefore, all such activity classes must be mapped into appropriate EmbeddedObjects following ALP structure and model definitions. Essential attributes of the mapping could be:

- **Duration.** The duration of an activity is the time interval between its start and end events. The duration is expressed in Java code as a ”duration” variable or ”durationFunc” of the aCTIVITY class (depending on how the duration is defined

in the source model). We can map the duration of an activity in the AnyLogic ALP file to the ”delayTime”-parameter of the corresponding EmbeddedObject.

- **Waiting timeout.** Waiting timeout is the time after which an enqueued activity will not be started, but removed from the queue. In Java code, it is defined in the ”waitingTimeout” attribute of aCTIVITY class. The corresponding attribute in the ALP file is ”timeout”-Parameter of the corresponding EmbeddedObject.

- **Resources**. Resource-constrained activities require certain resources to start, which are modeled by means of Resource Roles (with Resource Cardinality Constraints) and Resource Pools. They are defined in Java code as instances of the rESOURCErOLE class, which would be associated to the aCTIVITY class on a many-to-many basis. It is not trivial to map resource roles to specific elements in ALP file, but generally speaking, resource roles are turned into resource pool elements and referenced through the resourceSets/resourcePool parameters of the corresponding EmbeddedObject representing the resource-constrained activity. As shown in code example 5.3, the resource roles of an activity must be aligned to ”resourcePool”-elements in the ALP file.

The event flow relationship between activities of an AN is defined explicitly through ”successor activity”-attribute. Similarly, OESjava integrates the ”successorActivity”-attribute for each instance of the abstract class aCTIVITY (aCTIVITY.java), which is handled using the generic logic of Activity Networks included with the OES Core 2 simulator. In the case of the ALP file, one has to use the Connector element. Theoretically, given an activity class name and its successorActivity-attribute, one can program and map the corresponding connector element, where source is the activity class in consideration and target is its successor activity attribute.

5.4. Open issues

The main issue is that PN modeling tools, such as AnyLogic, do not support the general concept of activities (and Activity Networks), but only the more specific concept of processing activities. In addition to resource objects, a processing activity requires the participation of a processing object (”entity”). While processing nodes in a PN model have a location and a processing capacity (the number of processing objects that can be processed concurrently), nodes in an Activity Network, which represent basic activities, generally do not have either.

For example, in the AnyLogic Load-Haul-Dump model, trucks are objects that are being processed (thus doesn't require a separate resource pool), and the processing activity Load is performed at the processing node Load by a wheel loader (resource), that is processing a truck (entity). As a result, the AnyLogic process diagram's Load, Haul, Dump, and GoBack components do not represent activities but rather processing nodes. On the contrary, in the AnyLogic Make-and-Deliver Pizza model, an artificial abstract entity ”order” was introduced, since there are no entities that flow through the system

in the pizza service, but it is still required to have some "flowing entities" to be able to impose the PN paradigm.

As a result, such decisions regarding which objects will flow through the PN system should be made at the modeling stage with the assistance of modeling experts and cannot be resolved automatically using mapping/transformation tools. This aspect therefore complicates the process of developing unified/algorithmic mapping rules for objects across the two systems, because whereas processing entities are critical in the PN model, they are not as such in the AN model.

Another issue is that whenever the AnyLogic meta-model or syntax is changed with the new update, the corresponding mapping rules must be modified, since the mapping rules depend on domain-specific features in the meta-model.

5.5. Mapping rules

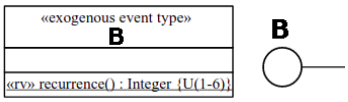
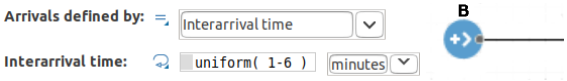
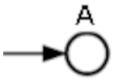
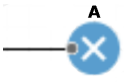
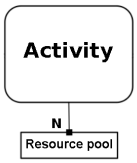
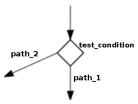
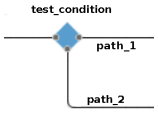
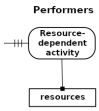
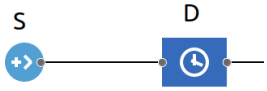
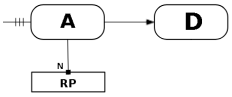
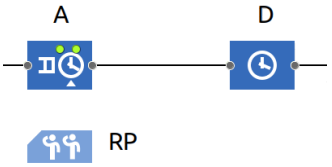
All that we are concerned about in this section is the mapping of formal models into their equivalents. A formal mapping requires the specification of models in well-defined modeling languages, as well as the definition of mapping rules in a model transformation language. It is possible to divide a mapping rule into two parts: a left-hand side that accesses the provided model, and a right-hand side that derives the target system. As a result, a model transformation is carried out in accordance with a clearly specified model transformation pattern. When it comes to maintaining model continuity, the target model should contain as much information from the source model as possible, and the initial modeling relationship should be maintained. [44].

The DPMN meta-model of the OES framework is located on the left-hand side of the table, while the AnyLogic meta-model is located on the right-hand side. Some mapping rules between the DPMN meta-model and AnyLogic meta-model are one-to-one mapping functions (resource pool, decision diamond), but some are not. There are also mapping rules, which are expressed in the form of patterns that relate to specific structures in each meta-model (e.g. "Resource-dependent activity with performer" <> "Service with resource pools").

Several mapping collisions were discovered during the rule's development, most notably regarding resource roles and resource pools. The OEM/DPMN, like BPMN, has resource roles for activities, which are defining properties for referencing the resources that are utilized. In the information model, resource roles and cardinality restrictions are specified, defining the resources that are necessary and acceptable. In contrast, the AnyLogic method defines resource responsibilities implicitly by associating resource pools (along with resource cardinality restrictions) with process model "blocks" such as Seize, Release, or Service. Similarly, the DPMN resource pool representation in AnyLogic corresponds to a resource pool element, resulting in a collision. AnyLogic's Seize-Delay-Release modeling method is procedural, whereas OEM/DPMN is more declarative.

Although most computational details, like the recurrence of exogenous events, the durations of activities, and all resource management items, such as resource roles for activity types, resource cardinality constraints, alternative resource types, and task pri-

orities, could be optionally displayed in an *enriched* DPMN process diagram, it is not possible to find a corresponding visual element in the AnyLogic meta-model, so most of these details are ignored and could be derived from the underlying OE class model.

Rule name	Left rule part	Right rule part
Exogenous event circle - Source		
Final event circle - Sink		
Resource role (with correspondent resource pool) - Resource pool		
Decision diamond - Select output		
Resource-dependent activity with performer - Service with resource pools		
Activity triggered by an event - Delay with incoming arrow from source		
Resource-constrained activity connected to other activity, which makes use of already allocated resources - Delay connected to service associated with resource pool		
Last event in the activity network - Seize/Service + Sink		

5.6. Transformation ideas

After mapping rules between two meta-models have been defined, it is possible to develop some concepts for a model transformation engine. The model transformation engine

executes the mapping rules on source models conforming to the DPMN/OEM meat-model and generates target models that conforms to the AnyLogic meta-model.

Because both the OES implementation and AnyLogic use Java as a host language, it is possible to integrate the same parsing and serialization libraries across both applications and obtain consistent results during model transportation and/or transformation. For the expression of the source model, the Extensible Markup Language (XML) could be used.

5.6.1. XML

XML is completely compatible with Java, a portable and extendable language that provides a readable and understandable format for humans and is well interpreted by machines.

XML is made up of two components: markup and content. The most often used construct is content, which is enclosed in markup by a start and an end tag.

```
<tagName>
   content
</tagName>
```

Listing 5.10: The XML content example

Together, the start tag, the content tag, and the appropriate end tag define an element. The content itself may contain one or more elements, referred to as child elements. This way, a hierarchical structure can be constructed by the use of nesting.

```
<tagName>
   <childTagName>
      childContent
   </childTagName>
</tagName>
```

Listing 5.11: The XML element's nesting example

The entire hierarchical structure is thus referred to as a tree, with the root element being the outermost element. Working with XML always involves walking through this tree structure, extracting, changing, or inserting items at the appropriate hierarchical level.

In particular, the Stax (Streaming API for XML) could be used as an API for reading and writing XML Documents. It was introduced in Java 6.0 and is considered superior to SAX and DOM [45].

5.6.2. Target XML file reconstruction/modification

One method how to modify the XML file containing the Anylogic model was proposed by authors in [46]. They have developed a Java program, that manipulates XML code to modify data on necessary abstract objects linked to blocks, such as connectors, sources,

and sinks. The Java program reads the template file's blocks and replicates them according to the supplied data. To add a new element (block) to the model, a node must be located in the XML tree that corresponds to a template block based on the attributes searched, then copy and connect the node to the original node's parent, and finally modify the data of the copied block (name of the block, position on the canvas, properties of the block, part of the programming code, etc.). The resulting XML structure is then saved to a new AnyLogic file.

The dissertation by Huang C. [47] also covers an example of a comparable transformation technique. Authors have defined the mapping rules between the SysML meta-model and the AnyLogic meta-model and exported the SysML model as an XMI file (sub type of XML file). The relevant elements in the resulting XMI file are then translated to their corresponding AnyLogic elements using Java code.

5.6.3. Dynamic instantiation

While it is possible to develop an AnyLogic model using XML files, it is not usually a good practice to do so by making direct adjustments to the ALP file itself. Instead, the one could use a Java library that reads XML files and in a startup code (for example triggered by the `OnStartup()` code) reason with the source XML file. The application will dynamically instantiate all kinds of objects specified in the model during the model runtime - walls, stations, delays, and so on. During the simulation run, a function with a loop walks the data and creates all the elements, initializes them, and makes them available for usage just as if they had been defined manually in the editor before.

Because the ALP file contains the source code, rather than the generated model, it is necessary to open AnyLogic after making direct modifications or recreation of the ALP file. Assuming that objects will be instantiated upon startup, the compiled AnyLogic model can be invoked from the command line or from other applications. Furthermore, the ALP file was not intended for direct editing, and new versions of AnyLogic are unlikely to be backwards compatible with previous versions of AnyLogic.

Notably, dynamically creating the space markup elements can be more time-consuming than creating the basic objects, resulting in a higher time-afford during development. In addition, developers must be prepared to invest time in error trapping bad inputs. Since in this approach we are crossing over into a file to build the model, we will need to make sure that the file does not have items that would cause the model to crash or violate underlying assumptions.

You might see the example of how to interpret XML data as input for AnyLogic in [49].

6. Summary and outlook

6.1. Summary

This thesis introduces a Java simulation framework for Object Event Simulation. This simulation framework consists of three main components: a set of basic classes for executing OE simulations, different simulation models (with tests) and a user interface. It is based on the OESjs simulation framework for activity-based simulation. The task was to reconstruct example simulation models in OESjava and AnyLogic on the basis of the DPMN process diagram, investigate core elements, and define the mapping between them.

The first part was the introduction to OEM&S - a new modeling and simulation paradigm with a formal semantics and an ontological foundation. The OEM has the OE/UML Class Diagrams as an information modeling language and DPMN (a variant of BPMN) as a process modeling language. Also, the extension of DPMN with Activity-Based Discrete Event Simulation (Activity Networks) is described, 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). It allows making visual object-oriented simulation design models that can be implemented with various (object-oriented) simulation technologies and model business processes with the help of activities. For the better understanding of the OES paradigm and the application of modeling languages required for it, it was necessary to introduce a simulation examples: the Make-and-Deliver Pizza and the Load-Haul-Dump models. Therefore, conceptual information and process models, in the form of OE class diagrams and BPMN/DPMN diagrams, respectively, were defined for each model. Afterwards, OE class diagrams were improved in information design, and process design models have been constructed.

The next task described in this thesis was the porting of OES Framework to Java. The OES has been implemented in JavaScript (OESjs), which is a dynamically-typed high-level coding language, whereby Java includes all concepts of object-oriented programming, type safety, specialized collections, etc. Some differences as well as similarities between Java and JavaScript were considered from a development perspective. The most interesting and challenging issues were described with the demonstration of solution approaches. Additionally, tools for developing and reconstruction of simulation logic have been outlined. A suitable UI to view simulation results has been developed. This reconstruction experience allowed us to evaluate the conformance and descriptive effectiveness of the given ontological model on the one hand, and the software portability of the OESjs framework on the other. Also, the obtained results may help to reconstruct the OES framework in other OO languages.

The next part of this work was investigation of the AnyLogic simulation platform to understand its process modeling concepts, and for comparing its DES approach with the business process modeling approach of DPMN. Before reconstructing the OES example models in AnyLogic, the main components of AnyLogic were illustrated and described. In addition, an attempt was made to analyze the structure and fundamental parts of the source files (having XML syntax) in which AnyLogic stores data about the simulation model. As the result of the investigation, semantics and modeling concepts of both AnyLogic and DPMN approaches were compared. After doing the investigation, it was discovered that the semantics and modeling ideas of both the AnyLogic and the DPMN approaches were markedly different from one another.

The last task of this thesis was to define the mapping rules between DPMN and AnyLogic diagram languages, as well as identify corresponding patterns between Java code and AnyLogic's ALP source file. In order to determine and prove these mapping rules the reconstructed Make-and-Deliver Pizza and Load-Haul-Dump models were used. Some ideas about how to apply resulting mapping rule for the developing of a transformation engine were given. Also an important feature was the comparison of business process models made with BPMN, which are essentially Activity Networks (AN) and the same business process models made with AnyLogic, which are essentially Processing Network (PN).

6.2. Possible further work

This paper has set the stage for further research and improvements in various fields. Among the possible topics for such research are the following:

- Development of a transformation engine between OESjava and AnyLogic that implements the supplied mapping rules. Chapter 5.6 makes some suggestions for how the transition might be accomplished.
- Integration of the processing network paradigm into the execution logic of the OESjava/OESjs framework, as well as the reconstruction of PN model examples. Additionally, the same models may be reconstructed in AnyLogic, and the current mapping rules may be updated accordingly. The issue of conceptual PN modeling with DPMN was already discussed by G. Wagner in [8].
- At the time of writing, OES was already implemented in JavaScript and Python. Likewise, the framework is now available in Java. The idea is to continuously add new OES implementations in all languages and then compare them all.
- Java is the host language used by AnyLogic, which allows the user to create their own Java classes in the model with any required functionality. This can be used to inject some OESjava code to provide a more precise simulation of activity network behavior.

Bibliography

[1] Gerd Wagner. Introduction to Simulation Using JavaScript. Proceedings of the 2016 Winter Simulation Conference. In: T.M.K. Roeder et al (eds.). Piscataway, New Jersey: IEEE.

[2] Gruber T. (2009) Ontology. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA.
`https://doi.org/10.1007/978-0-387-39940-9_1318`

accessed 18^{th} September 2021.

[3] Core Simulators for Object Event Simulation
`https://github.com/gwagner57/oes`

accessed 16^{th} July 2021.

[4] Prof. Gerd Wagner (SIMULTECH 2020). *"Object Event Simulation"*.
`https://dpmn.info/reading/SIMULTECH2020/`

accessed 25^{th} July 2021.

[5] Gordon G. *A General Purpose Systems Simulation Program.* // McMillan NY, Proceedings of EJCC, Washington D.C., 1961.

[6] Wagner, G. 2021. *Discrete Event Simulation Engineering.*
`https://sim4edu.com/reading/des-engineering/`

accessed 25^{th} July 2021.

[7] Wagner, G. 2020. "Business Process Modeling and Simulation with DPMN: Resource-Constrained Activities". In Proceedings of the 2020 Winter Simulation Conference, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing. 762?773. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

[8] Wagner, G. 2019. "Information and Process Modeling for Simulation ? Part II: Activities and Processing Networks".
`https://dpmn.info/reading/Activities.html`

accessed 25^{th} July 2021.

[9] Wagner, G. 2018. "Information and Process Modeling for Simulation ? Part I: Objects and Events". Journal of Simulation Engineering 1:1?25.

[10] Guizzardi, G., & Wagner, G. (2013). Dispositions and Causal Laws as the Ontological Foundation of Transition Rules in Simulation Models. In Proceedings of the 2013 Winter Simulation Conference (pp. 1335?1346). Piscataway, NJ: IEEE

[11] Buss, Arnold. *Basic Event Graph Modeling*. Simulation News Europe, Technical Notes, April 2001.

[12] What is Java technology?
`https://www.java.com/en/download/help`
accessed 17^{th} July 2021.

[13] Schildt, Herbert. Java. 2019.

[14] Vernon-Bido, Daniele and Collins, Andrew and Sokolowski, John. (2015). Effective visualization in modeling and simulation. Simulation Series. 47. 33-40.

[15] Graphviz
`https://www.graphviz.org/`
accessed 17^{th} July 2021.

[16] Java 3D API
`https://www.oracle.com/java/technologies/javase/java-3d.html`
accessed 16^{th} July 2021.

[17] Raff, Edward. (2017). JSAT: Java Statistical Analysis Tool, a Library for Machine Learning. Journal of Machine Learning Research. 23. 1–5.
`http://jmlr.org/papers/v18/16-131.html`
accessed 19^{th} July 2021.

[18] List of discrete event simulation software
`https://en.wikipedia.org/wiki/List_of_discrete_event_simulation_software`
accessed 19^{th} July 2021.

[19] History of Spring and the Spring Framework
`https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html`
accessed 19^{th} July 2021.

[20] Spring boot
`https://spring.io/projects/spring-boot`
accessed 21^{th} July 2021.

[21] Spring dominates the Java ecosystem with 60% using it for their main applications
`https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/`
accessed 21^{th} July 2021.

[22] What is PostgreSQL?
https://www.postgresql.org/about/
accessed 21th July 2021.

[23] Docker overview
https://docs.docker.com/get-started/overview/
accessed 21th July 2021.

[24] Tutorial: Using Thymeleaf. Document version: 20181029 - 29 October 2018.
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
accessed 21th July 2021.

[25] Apache Maven
https://maven.apache.org/
accessed 22th July 2021.

[26] What Is JUnit Testing?
https://www.softwaretestinghelp.com/junit-tutorial/
accessed 29th August 2021.

[27] MDN Web Docs. "Object.prototype.constructor".
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/constructor
accessed 23th July 2021.

[28] Hautus, E.. ?IMPROVING JAVA SOFTWARE THROUGH PACKAGE STRUCTURE ANALYSIS.? (2002).

[29] Oracle documentation. Class Constructor<T>.
https://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Constructor.html
accessed 31th July 2021.

[30] OES JavaScript Core Simulators
https://gwagner57.github.io/oes/js/index.html
accessed 1st August 2021.

[31] Using Global Variables/Constants in Java.
https://stackabuse.com/using-global-variables-constants-in-java
accessed 1st August 2021.

[32] Class Class in Java
https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html
accessed 1st August 2021.

[33] forName method in Java
`https://developer.android.com/reference/java/lang/Class#forName`
accessed 1^{st} August 2021.

[34] Inventory Management with a Continuous Replenishment Policy simulation scenario.
`https://sim4edu.com/sims/4/description`

[35] AnyLogic 8 Documentation `https://help.anylogic.ru/index.jsp` accessed 10^{th} September 2021.

[36] Grigoryev, Ilya. AnyLogic 7 in Three Days: A Quick Course in Simulation Modeling. 2015.

[37] AnyLogic Library Reference Guides
`https://anylogic.help/library-reference-guides/process-modeling-library`
accessed 11^{th} August 2021.

[38] Network in AnyLogic
`https://anylogic.help/markup/network.html`
accessed 18^{th} August 2021.

[39] Event scheduling in AnyLogic
`https://anylogic.help/anylogic/statecharts/event-scheduling.html`
accessed 18^{th} August 2021.

[40] Agent based simulation modeling in AnyLogic
`https://www.anylogic.com/use-of-simulation/agent-based-modeling/`
accessed 18^{th} August 2021.

[41] Hoelldobler, Katrin & Roth, Alexander & Rumpe, Bernhard & Wortmann, Andreas. (2017). *Advances in Modeling Language Engineering*. 3-17. 10.1007/978-3-319-66854-3_1.

[42] Deniz Çetinkaya, Alexander Verbraeck, and Mamadou D. Seck. 2015. Model Continuity in Discrete Event Simulation: A Framework for Model-Driven Development of Simulation Models. ACM Trans. Model. Comput. Simul. 25, 3, Article 17 (May 2015), 24 pages. DOI:https://doi.org/10.1145/2699714

[43] Hartmut Ehrig and Claudia Ermel. 2008. Semantical correctness and completeness of model transformations using graph and rule transformation. In Proceedings of the 4th International Conference on Graph Transformations (ICGT?08). 194?210.

[44] Ehrig H., Ermel C. (2008) Semantical Correctness and Completeness of Model Transformations Using Graph and Rule Transformation. In: Ehrig H., Heckel R., Rozenberg G., Taentzer G. (eds) Graph Transformations. ICGT 2008. Lecture Notes in Computer Science, vol 5214. Springer, Berlin, Heidelberg.

[45] StAX (Streaming API for XML).
`https://docs.oracle.com/javase/tutorial/jaxp/stax/index.html`
accessed 21^{st} August 2021

[46] Rodic, Blaz. (2017). Industry 4.0 and the New Simulation Modelling Paradigm. Organizacija. 50. 10.1515/orga-2017-0017.

[47] Huang, C. (2011). Discrete event system modeling using SysML and model transformation.

[48] Borshchev, A. (2013). The big book of simulation modeling: Multimethod modeling with AnyLogic 6. Lisle, IL: AnyLogic North America.

[49] Example: XML with DOM in AnyLogic
`https://cloud.anylogic.com/model/330bae1c-5161-4dfa-b22f-fa46c374c18d?mode=SETTINGS`
accessed 30^{th} August 2021

Appendices

A. Appendix

A.1. Docker file

```
FROM postgres
ENV POSTGRES_USER oes
ENV POSTGRES_PASSWORD oes
ENV POSTGRES_DB oes
```

A.2. Content of pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
       https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.4.1</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>de.oes</groupId>
   <artifactId>core2</artifactId>
   <version>0.0.2-SNAPSHOT</version>
   <packaging>war</packaging>
   <name>core2</name>
   <description>Core 2 Simulators for Object Event Simulation</description>
   <properties>
      <java.version>11</java.version>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-thymeleaf</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.vladmihalcea</groupId>
    <artifactId>hibernate-types-52</artifactId>
    <version>2.10.4</version>
</dependency>
<dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <optional>true</optional>
</dependency>
<dependency>
   <groupId>javax.validation</groupId>
   <artifactId>validation-api</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>
<dependency>
      <groupId>javax.persistence</groupId>
      <artifactId>javax.persistence-api</artifactId>
  </dependency>
<dependency>
   <groupId>org.hibernate.validator</groupId>
   <artifactId>hibernate-validator</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-tomcat</artifactId>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>org.postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-math3</artifactId>
```

```
        <version>3.6.1</version>
      </dependency>
   </dependencies>

   <build>
    <plugins>
      <plugin>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-maven-plugin</artifactId>
         <configuration>
            <excludes>
               <exclude>
                  <groupId>org.projectlombok</groupId>
                  <artifactId>lombok</artifactId>
               </exclude>
            </excludes>
         </configuration>
      </plugin>
   </plugins>
   </build>

</project>
```

A.3. Content of loadhauldump.xml

```
<!doctype html>
<html lang="en">
<head>
 <style>
      table,
      th,
      td {
        padding: 10px;
        border: 1px solid black;
        border-collapse: collapse;
      }
    </style>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<title>OES Core 2 Simulation Examples</title>
<meta name="viewport" content="width=device-width, initial-scale = 1.0" />
<meta name="description"
   content="Simulation examples for OES Core simulators" />
<style>
p.description {
   font-size: smaller;
}
```

```
</style>
</head>
<body>

   <main>
      <h1>Load-Haul-Dump Simulation</h1>
      <p th:text="${expInfo}"></p>

<!--  EXPERIMENT TABLE -->
   <table th:if="${type} == 1">
      <thead>
         <tr>
            <th>Key</th>
            <th>Value</th>
         </tr>
      </thead>
      <tbody>
         <tr th:each="entry : ${stat.getExperiments()}">
            <td th:text="${entry.key}"></td>
            <td th:text="${entry.value}"></td>
         </tr>
         <tr th:each="entry : ${stat.getSumStat()}">
            <td th:text="${entry.key}"></td>
            <td th:text="${entry.value}"></td>
         </tr>
      </tbody>
   </table>

<!--  SIMPLE STAT TABLE -->
      <table th:if="${type} == 0">
         <thead>
            <tr>
               <th>Key</th>
               <th>Value</th>
            </tr>
         </thead>
         <tbody>
            <tr th:each="entry : ${stat.entrySet()}">
               <td th:text="${entry.key}"></td>
               <td th:text="${entry.value}"></td>
            </tr>
         </tbody>
      </table>

<!-- ACTIVITY STAT TABLE -->
   <table th:if="${type} == 0">
      <thead>
           <tr>
              <th>Key</th>
```

```
      <th>Value</th>
   </tr>
</thead>
<tbody>
   <tr th:each="entry : ${actStat.entrySet()}">
      <td th:text="${entry.key}"></td>
      <td>
         <table>
            <tr>
                  <td>enqueued activities</td>
                  <td
                     th:text="${entry.value.getEnqueuedActivities()}"></td>
             </tr>
             <tr>
                  <td>started activities</td>
                  <td
                     th:text="${entry.value.getStartedActivities()}"></td>
             </tr>
             <tr>
                  <td>completed activities</td>
                  <td
                     th:text="${entry.value.getCompletedActivities()}"></td>
             </tr>
             <tr>
                  <td>waiting time</td>
                  <td
                     th:text="${entry.value.getWaitingTime().getMax()}"></td>
             </tr>
              <tr>
                  <td>cycle time</td>
                  <td
                     th:text="${entry.value.getCycleTime().getMax()}"></td>
             </tr>
             <tr>
                  <td>queue length</td>
                  <td
                     th:text="${entry.value.getQueueLength().getMax()}"></td>
             </tr>
             <tr>
                  <td>res util</td>
                  <td>
                     <table>
                        <tr th:each="e : ${entry.value.getResUtil()}">
                           <td th:text="${e.key}"></td>
                           <td th:text="${e.value}"></td>
                        </tr>
                     </table>
                  </td>
               </tr>
```

```
                </table>
            </tr>
        </tbody>
    </table>

    <table th:if="${hasLogs}">
        <thead>
            <tr>
                <th>Step</th>
                <th>Time</th>
                <th>System State</th>
                <th>Future Events</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="entry : ${logs}">
                <td th:text="${entry.getSimStep()}"></td>
                <td th:text="${entry.getTime()}"></td>
                <td th:text="${entry.getSystemState()}"></td>
                <td th:text="${entry.getFutureEvents()}"></td>
            </tr>
        </tbody>
    </table>


  </main>
  <footer>
    <p>
      2021 G. Wagner (<a
        href="https://creativecommons.org/licenses/by/4.0/">CC BY</a>),
      Brandenburg University of Technology, Germany
    </p>
  </footer>
</body>
</html>
```

A.4. Content of index.xml

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<title>OES Core Simulation Examples</title>
<meta name="viewport" content="width=device-width, initial-scale = 1.0" />
<meta name="description"
  content="Simulation examples for OES Core simulators" />
```

```
<style>
p.description {
   font-size: smaller;
}
</style>
</head>
<body>

   <main>
      <h1>Simulation Examples</h1>
      <h2>OESjs Core 2</h2>
      <p class="description">The OESjs Core 2 simulator implements an
          architecture for Object Event Simulation (OES) that extends the OESjs
          Core 1 simulator by adding support for activities, which may be
          resource-constrained.</p>
      <nav>
         <ul>
            <li><a href="/core2/pizzaservice1">Pizza-Service-1</a></li>
            <li><a href="/core2/pizzaservice2">Pizza-Service-2</a></li>
            <li><a href="/core2/makeanddeliver">Make-and-Deliver</a></li>
            <li><a href="/core2/loadhauldump">Load-Haul-Dump</a></li>

            <li><a href="/core2/medical-department-1a">Medical-department-1a</a>:
         <p> This example explicitly models the resource management of doctors
             with the help of a counter variable for available doctors in the
             form of an attribute nmrOfAvailableDoctors,
         and the operations isDoctorAvailable(), allocateDoctor() and
             releaseDoctor(), in the MedicalDepartment class. </p> </li>

         <li><a href="/core2/medical-department-1b">Medical-department-1b</a>:
         <p> This model is essentially the same as Medical-Department-1a,
             except that it uses the generic resource management logic that is
             built into OES Core 2.
         The resource pool doctors is modeled as a count pool, which abstracts
             away from individual resources and only counts them. </p> </li>

         <li><a href="/core2/medical-department-1c">Medical-department-1c</a>:
         <p> In this model, the resource pool doctors is modeled as an
             individual resource pool instead of a count pool.
         This allows making the model more realistic, for instance, by
             assigning an individual work schedule to each doctor defining her
             availability. </p> </li>

            <li><a href="/core2/medical-department-2a">Medical-department-2a</a>:
         <p> This model includes two activity types: WalkToRoom activities
             involve a room and are performed by a nurse,
         while Examination activities involve a room and are performed by a
             doctor.
```

```
        The resource pools nurses and doctors are modeled as individual
            resource pools, while the resource pool rooms,
        which is used by both WalkToRoom and Examination activities, is
            modeled as a count pool. </p> </li>

        <li><a href="/core2/medical-department-2b">Medical-department-2b</a>:
        <p> This model is the same as Medical-Department-2a, except that
            Examination activities require a room, two nurses and a doctor.
            </p> </li>
        </ul>
      </nav>
   </main>
   <footer>
      <p>
         2021 G. Wagner (<a
            href="https://creativecommons.org/licenses/by/4.0/">CC BY</a>),
         Brandenburg University of Technology, Germany
      </p>
   </footer>
</body>
</html>
```

A.5. Resulting statistics for running 10 replications of the Inventory-Management-Scenario

Key	Value
0	{nmrOfStockOuts=16.0, serviceLevel=98.4, lostSales=117.0}
1	{nmrOfStockOuts=22.0, serviceLevel=97.8, lostSales=172.0}
2	{nmrOfStockOuts=22.0, serviceLevel=97.8, lostSales=151.0}
3	{nmrOfStockOuts=14.0, serviceLevel=98.6, lostSales=80.0}
4	{nmrOfStockOuts=15.0, serviceLevel=98.5, lostSales=112.0}
5	{nmrOfStockOuts=17.0, serviceLevel=98.3, lostSales=168.0}
6	{nmrOfStockOuts=25.0, serviceLevel=97.5, lostSales=182.0}
7	{nmrOfStockOuts=18.0, serviceLevel=98.2, lostSales=154.0}
8	{nmrOfStockOuts=12.0, serviceLevel=98.8, lostSales=109.0}
9	{nmrOfStockOuts=15.0, serviceLevel=98.5, lostSales=167.0}
lost sales	{Minimum=80.0, Maximum=182.0, Std.dev.=32.37, CI Lower=121.14, Average=141.2, CI Upper=161.26}
nmrOfStockOuts	{Minimum=12.0, Maximum=25.0, Std.dev.=3.93, CI Lower=15.16, Average=17.6, CI Upper=20.04}
serviceLevel	{Minimum=97.5, Maximum=98.8, Std.dev.=0.39, CI Lower=97.42, Average=98.24, CI Upper=97.98}

A.6. Content of RunLoadHaulDumpSimulationService.java

```
package de.oes.core2.endpoint.service;

@Component
public class RunLoadHaulDumpSimulationService {

   /**
```

```
 * Runs the load-haul-dump simualtion model
 * @param dto - simulation settings defined by client
 * @param m - Thymeleaf model v
 */
public void run(SimulationSettingsDTO dto, org.springframework.ui.Model m) {
   Model model = initializeModel();
   Simulator sim = new Simulator();
   Scenario scenario = new Scenario();

   if(dto.getInit() == 0) { // (0) Basic scenario
      scenario = initScenario(sim);
   } else { // (1) Model variant
      scenario = initAltScenario(sim);
   }

   setStatisticVariables(model, sim);

   initSimulation(model, sim, scenario);

   if(dto.getType() == 0) { // (0) Standalone scenario
      runStandaloneScenario(sim, dto.isSimulationLog());
      m.addAttribute("stat", sim.getStat().getSimpleStat());
      calculateResUtil(sim.getStat().getActTypes().values(), sim);
      m.addAttribute("actStat", sim.getStat().getActTypes());
      if(dto.isSimulationLog()) m.addAttribute("logs",
          SimulatorLogs.getLogs());
   } else { // (1) Simple Experiment with 10 replications, each running for
       1000 min.
      eXPERIMENTtYPE expType = defineExperimentType(model, scenario);
      ExperimentsStatisticsDTO resutlDTO = runExperiment(sim, expType,
          dto.isSimulationLog());
      m.addAttribute("stat", resutlDTO);
   }
}

/**
 * Calculate resource utilization for activity types
 * @param activityStats - collected activity statistics
 * @param sim - current simulation
 */
private void calculateResUtil(Collection<ActivityStat> activityStats,
    Simulator sim) {
   for (ActivityStat activityStat : activityStats) {
      activityStat.getResUtil().replaceAll((k,v) ->
          MathLib.round(v.doubleValue() / sim.getTime()));
   }
}

/**
```

```
 * Runs simulation scenario with replications (as experiment)
 * @param sim - current simulation
 * @param expType - type of the experiment
 * @param logs - defines if simulation steps must be logged
 * @return
 */
private ExperimentsStatisticsDTO runExperiment(Simulator sim, eXPERIMENTtYPE
    expType, boolean logs) {
   autowireCapableBeanFactory.autowireBean(sim);
   sim.setExperimentType(expType);
   return sim.runExperiment(logs);
}

/**
 * Run simulation as a simple scenario without replications
 * @param sim - current simulation
 * @param logs - defines if simulation steps must be logged
 */
private void runStandaloneScenario(Simulator sim, boolean logs) {
   sim.runStandaloneScenario(logs);
   System.out.println("FINAL STAT");
   for (Entry<String, Number> e : sim.getStat().getSimpleStat().entrySet()) {
      System.out.println(e.getKey() + " : " + e.getValue());
   }
}

/**
 * Set simualtion model and simulation scenario for the simulator
 * @param model - simulation model
 * @param sim - simulator
 * @param scenario - simulation scenario
 */
private void initSimulation(Model model, Simulator sim, Scenario scenario) {
   sim.setModel(model);
   sim.setScenario(scenario);
}

/**
 * Definition of experiment type for the given model
 * @param model - simulation model
 * @param scenario - current simulation scenario
 * @return
 */
private eXPERIMENTtYPE defineExperimentType(Model model, Scenario scenario) {
   eXPERIMENTtYPE expType = new eXPERIMENTtYPE(
      model,
      "Simple Experiment with 10 replications, each running for " +
          scenario.getDurationInSimTime() + " " + model.getTimeUnit(),
      10, // nmrOfReplications
```

```
      null, // parameterDef
      new Integer[] {123, 234, 345, 456, 567, 678, 789, 890, 901, 1012} //
          seeds
      );
   expType.setStoreExpResults(true);
   return expType;
}

/**
 * Initialization of statistic variables
 * @param model - simulation model
 * @param sim - current simulation
 */
private void setStatisticVariables(Model model, Simulator sim) {
   Consumer<Simulator> setupStatistics = s -> {
   };
   model.setSetupStatistics(setupStatistics);
}

/**
 * Initialization of alternative simulation scenario
 * @param sim - current simulation
 * @return initialized simulation scenario object
 */
public Scenario initAltScenario(Simulator sim) {
   sim.getAClasses().put("GoToLoadingSite", new
       GoToLoadingSite(sim,0,0,null));
   sim.getAClasses().put("Load", new Load(sim,0,0,null));
   sim.getAClasses().put("Haul", new Haul(sim,0,0,null));
   sim.getAClasses().put("GoBackToLoadingSite", new
       GoBackToLoadingSite(sim,0,0,null));
   sim.getAClasses().put("GoHome", new GoHome(sim,0,0,null));
   sim.getAClasses().put("Dump", new Dump(sim,0,0,null));
   Scenario altScenario = new Scenario();
   altScenario.setScenarioNo(1l);
   altScenario.setTitle("Scenario with 2 wheel loaders");
   altScenario.setDescription(
      "Based on the default scenario (with 5 trucks and 1 wheel loader), "
   + "this scenario has a second wheel loader. As a consequence, <i>Load</i>
       activities are performed twice as fast.");
   Consumer<Simulator> setupInitialStateAlt = s -> {
       // Create initial objects
      Truck t1 = new Truck(1, "t1", sim, rESOURCEsTATUS.AVAILABLE);
      Truck t2 = new Truck(2, "t2", sim, rESOURCEsTATUS.AVAILABLE);
      Truck t3 = new Truck(3, "t3", sim, rESOURCEsTATUS.AVAILABLE);
      Truck t4 = new Truck(4, "t4", sim, rESOURCEsTATUS.AVAILABLE);
      Truck t5 = new Truck(5, "t5", sim, rESOURCEsTATUS.AVAILABLE);
      WheelLoader wl1 = new WheelLoader(11, "wl1", sim,
          rESOURCEsTATUS.AVAILABLE);
```

```
        WheelLoader wl2 = new WheelLoader(12, "wl2", sim,
            rESOURCEsTATUS.AVAILABLE);
        // Initialize the individual resource pools
        rANGE range = new rANGE();

        rESOURCEpOOL rp1 = new rESOURCEpOOL(s, "trucks", range, 5, List.of(t1,
            t2, t3, t4, t5));
        t1.setResourcePool(rp1);
        t2.setResourcePool(rp1);
        t3.setResourcePool(rp1);
        t4.setResourcePool(rp1);
        t5.setResourcePool(rp1);

        rESOURCEpOOL rp2 = new rESOURCEpOOL(s, "wheelLoaders", range, 2,
            List.of(wl1, wl2));
        wl1.setResourcePool(rp2);
        wl2.setResourcePool(rp2);

        s.getResourcepools().put("trucks", rp1);
        s.getResourcepools().put("wheelLoaders", rp2);

        GoToLoadingSite.resRoles.get("trucks").setResPool(rp1);
        Load.resRoles.get("trucks").setResPool(rp1);
        Load.resRoles.get("wheelLoaders").setResPool(rp2);
        Haul.resRoles.get("trucks").setResPool(rp1);
        GoHome.resRoles.get("trucks").setResPool(rp1);
        Dump.resRoles.get("trucks").setResPool(rp1);
        GoBackToLoadingSite.resRoles.get("trucks").setResPool(rp1);
        // Schedule initial events
        s.getFEL().add(new HaulRequest(s, 1l, null, 990));

    };
    altScenario.setSetupInitialState(setupInitialStateAlt);
    sim.getScenarios().add(altScenario);
    return altScenario;
}

/**
 * Initialization of a basic simulation scenario
 * @param sim - current simulation
 * @return initialized simulation scenario object
 */
private Scenario initScenario(Simulator sim) {
    sim.getAClasses().put("GoToLoadingSite", new
        GoToLoadingSite(sim,0,0,null));
    sim.getAClasses().put("Load", new Load(sim,0,0,null));
    sim.getAClasses().put("Haul", new Haul(sim,0,0,null));
    sim.getAClasses().put("GoBackToLoadingSite", new
        GoBackToLoadingSite(sim,0,0,null));
```

```
    sim.getAClasses().put("GoHome", new GoHome(sim,0,0,null));
    sim.getAClasses().put("Dump", new Dump(sim,0,0,null));
    Scenario scenario = new Scenario();
    scenario.setTitle("The default scenario has 5 trucks (with IDs 1-5) and
        one wheel loader (with ID 11).");
    // Initial State
    Consumer<Simulator> setupInitialState = s -> {
        // Create initial objects
       Truck t1 = new Truck(1, "t1", sim, rESOURCEsTATUS.AVAILABLE);
       Truck t2 = new Truck(2, "t2", sim, rESOURCEsTATUS.AVAILABLE);
       Truck t3 = new Truck(3, "t3", sim, rESOURCEsTATUS.AVAILABLE);
       Truck t4 = new Truck(4, "t4", sim, rESOURCEsTATUS.AVAILABLE);
       Truck t5 = new Truck(5, "t5", sim, rESOURCEsTATUS.AVAILABLE);
       WheelLoader wl1 = new WheelLoader(11, "wl1", sim,
           rESOURCEsTATUS.AVAILABLE);
        // Initialize the individual resource pools
       rANGE range = new rANGE();

       rESOURCEpOOL rp1 = new rESOURCEpOOL(s, "trucks", range, 5, List.of(t1,
           t2, t3, t4, t5));
       t1.setResourcePool(rp1);
       t2.setResourcePool(rp1);
       t3.setResourcePool(rp1);
       t4.setResourcePool(rp1);
       t5.setResourcePool(rp1);

       rESOURCEpOOL rp2 = new rESOURCEpOOL(s, "wheelLoaders", range, 1,
           List.of(wl1));
       wl1.setResourcePool(rp2);

       s.getResourcepools().put("trucks", rp1);
       s.getResourcepools().put("wheelLoaders", rp2);

       GoToLoadingSite.resRoles.get("trucks").setResPool(rp1);
       Load.resRoles.get("trucks").setResPool(rp1);
       Load.resRoles.get("wheelLoaders").setResPool(rp2);
       Haul.resRoles.get("trucks").setResPool(rp1);
       GoHome.resRoles.get("trucks").setResPool(rp1);
       Dump.resRoles.get("trucks").setResPool(rp1);
       GoBackToLoadingSite.resRoles.get("trucks").setResPool(rp1);
       // Schedule initial events
       s.getFEL().add(new HaulRequest(s, 1l, null, 990));

    };
    scenario.setSetupInitialState(setupInitialState);
    return scenario;
}

/**
```

```
     * Initialization of simulation model parameters
     * @return initialized simulation model object
     */
    private Model initializeModel() {
        Model model = new Model();
        model.setName("Load-Haul-Dump-1");
        model.setTime(Time.CONT);
        model.setTimeUnit(TimeUnit.min);
        model.setObjectTypes(List.of(Truck.class, WheelLoader.class));
        model.setEventTypes(List.of(HaulRequest.class));
        model.setActivityTypes(Set.of("GoToLoadingSite","Load",
            "Haul","Dump","GoBackToLoadingSite","GoHome"));
        return model;
    }
}
```

B. Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterarbeit selbständig und ohne unerlaubte Hilfe angefertigt habe, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Cottbus, den _____ Unterschrift: _____