Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 200 (2022) 386–393

Procedia Computer Science

www.elsevier.com/locate/procedia

3rd International Conference on Industry 4.0 and Smart Manufacturing

A brief introduction to deploy Amazon Web Services for online discrete-event simulation

Wladimir Hofmann[a], Sebastian Lang[b,c,*], Paul Reichardt[c] ,Tobias Reggelin[b,c]

[a]*Kühnehöfe 33, 22761 Hamburg, Germany*
[b]*Fraunhofer Institute for Factory Operation and Automation IFF, Sandtorstr. 22, 39106 Magdeburg, Germany*
[c]*Otto von Guericke University Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany*

Abstract

The concept of digital twin demands fast and efficient methods to distribute, initialize, and execute simulation models that replicate real-world systems and processes. In the field of production and logistics, discrete-event simulation (DES) is the method of choice when it comes to developing digital twins. Cloud platforms such as Amazon Web Services (AWS) give the opportunity to distribute and parallelize the computationally expensive execution of simulation models on high-performance systems. However, there is presently no guidance on the real-time deployment of DES models on cloud infrastructures, making the topic difficult to approach. Against this background, we present an intuitive and practice-oriented procedure model for deploying DES models on AWS. We support our explanations with a demonstration of our procedure model using an end-to-end example of a simple order manufacturing process. Get the repository from GitHub: https://github.com/fladdimir/csa-simulation-based-sc-forecast

Keywords: Amazon Web Services; AWS; Discrete-Event Simulation; DES; Cloud Simulation; Online Simulation; Real-Time; Digital Twin

1. Introduction

Discrete-event simulation (DES) is an established method for analyzing, planning and optimizing processes and systems of production and logistics. DES models can be integrated with a variety of applications, for instance, with

* Corresponding author. Tel.: +49-391-67-57299 ; fax: +49-391-67-42646.
E-mail address: sebastian.lang@iff.fraunhofer.de

1877-0509
10.1016/j.procs.2022.01.237

heuristics and metaheuristics to solve combinatorial optimization problems or with information systems to serve as a digital twin of the real system. The main benefit of DES models is that they can represent processes and systems in an arbitrary level of detail. On the other hand, the execution of complex models can be computationally costly. In this case, for instance, simulation-based optimization, which usually requires a large number of simulation experiments, takes a considerable amount of time.

Cloud computing services, such as Amazon Web Services, Microsoft Azure or Google Cloud, could contribute to addressing this challenge, as they allow an individually scalable parallelization of simulation experiments. Furthermore, simulation processes of very extensive and detailed models can be distributed to different resources and sub-processes [1]. Such cloud infrastructures could be especially attractive for small and medium-sized enterprises, for which the utilization of cloud computing services based on a pay-per-use model is more economical than the investment and maintenance of a high-performance workstation or cluster.

1.1. Related research

The research on cloud-based deployment of DES models is still in its infancy. Frequently discussed problems are performance issues and deadlocks arising when the execution of a DES model is distributed among different cloud resources. In particular, those problems are likely to occur if the simulation of one subprocess causes subsequent events in another subprocess simulated on a different resource. Due to different event lists, the simulation execution of different subprocesses is asynchronous by nature. Thus, at the same point in time, the virtual simulation time of a subprocess *A* may be further advanced than that of another subprocess *B*. Let us assume that subprocess *B* schedules a new event in subprocess *A* at a point in time that is already in *A*'s past. Such an event violates the so-called Local Causality Constraint, which requires that incoming events are processed in ascending order according to the scheduled timestamps [2]. Against this background, some articles investigate different synchronization strategies for cloud-based distribution and parallelized execution of DES models [3; 4]. Javer et al. present an extensive literature review on this topic [5]. Vanmechelen et al. evaluate the Amazon cloud service EC2 for the distribution and parallelization of DES models [6]. The authors analyze different synchronization strategies in terms of runtime and event-processing rate. The results show that there are only insignificant performance deviations compared to the distribution and parallelization on a local cluster.

The research papers discussed in the first paragraph primarily address the correct and efficient parallelization of distributed DES models on cloud infrastructures. In contrast, some articles present new methods and tools that favor a cloud-based deployment of DES models. Liu et al. explore how DES models must be designed to run efficiently on the cloud [7]. In particular, the authors present a requirement analysis, a new architecture as well as new methods for the cloud-based deployment of DES models, and further a general procedure model for migrating existing DES tools to cloud infrastructures. Two other articles describe DES tools that are specifically tailored to be operated on cloud infrastructures. Heavey et al. present ManPy [8], a Python-based DES library whose components are distributed on a dedicated server architecture. Padilla et al. present a Java-based DES tool, which runs on Amazon Web Services cloud infrastructure [9]. Both simulators offer fully web-based modeling. Meanwhile, some commercial DES tools also support a cloud-based execution of models, e.g. AnyLogic [10].

1.2. Objective and Contribution

In summary, the recent state of research primarily addresses synchronization methods, which are particularly relevant for the distributed simulation of DES models. Two other articles present cloud-enabled DES tools. Only a single article describes strategies for the design of DES models that can be efficiently distributed and parallelized on cloud infrastructures. To the best of our knowledge, there are no publications to date that present an intuitive and practice-oriented procedure model for deploying DES models on established cloud computing platforms. The present work contributes to closing this gap.

Using Amazon Web Services (AWS) as an example, this paper presents a simple procedure model that allows an efficient distribution of DES models on the AWS cloud. We tailor the procedure model to a single cloud computing service provider, in order to make the technical implementation as illustrative as possible. The decision for AWS is

justified by the fact that Amazon currently owns the largest market share of all cloud computing service providers [11].

2. A procedure model for the deployment of DES models on the AWS cloud

The design of the procedure is subject to the requirement that a DES model deployed on AWS cloud can be used as a digital twins of a real system. Therefore, the DES model must be able to synchronize state changes in the real system in real-time. Considering a continuous input data stream, this requires especially a fast initialization and execution of the DES model. Against this background, the design of the procedure is based on the PhD thesis of Donhauser [12], describing a method for order control in hybrid production systems that is essentially based on an online simulation. Fig. 1 shows the procedure model including all necessary AWS services for the online execution of DES models on the cloud. In the following, we will explain each step in detail.

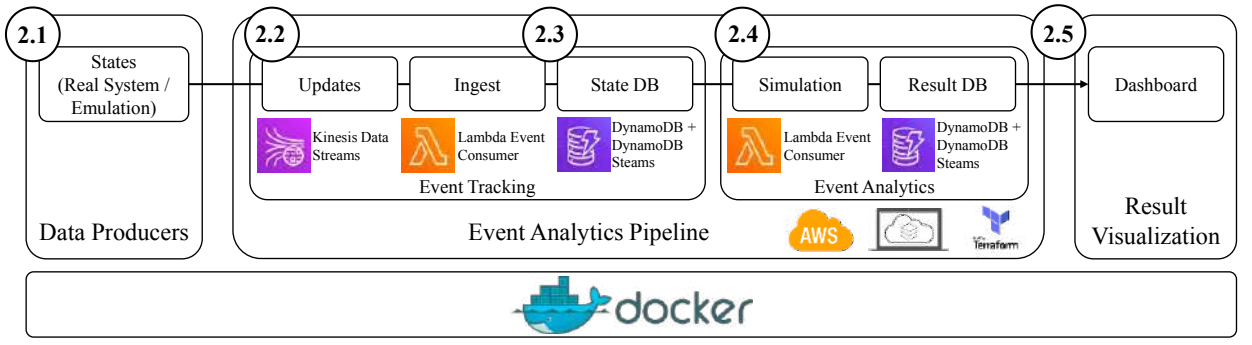

Fig. 1. Procedure model for online deployment of DES models on the AWS cloud.

2.1. Incoming events of data producers

Starting point is a process that acts as a data producer. The process is either embedded in a physical system (e.g., production system) that collects information via sensors and other IoT devices, or represented by a service that provides real (e.g., information system) or synthetic information (e.g., DES model). At each point in time when a measurement takes place, the measured state will be broadcast as an event.

2.2. Capturing events with AWS Kinesis

We use the AWS service *Kinesis* to buffer incoming information for further processing. Kinesis acts as a communication interface and decouples the data generating process from the data consuming DES model. In this way, state changes that occur during the execution of the DES model are not being lost. A Kinesis stream consists of one or more shards that manage sequences of data. A shard can read up to five data transactions per second or a data volume of 2 MB per second, and write up to 1,000 data transactions per second or a data volume of 1 MB per second. Kinesis puts each data transaction into one shard based on its specified partition key value. The correct processing of data transactions is guaranteed only on partition key level. The capacity of a Kinesis data stream can be customized via the number of shards.

2.3. Processing events with AWS Lambda and persisting system states with DynamoDB

The state events buffered in the Kinesis data stream are processed FIFO and written to a state database connected to the DES model. We implement the database with the AWS service *DynamoDB*, which provides a fast, flexible and NoSQL managed database framework. Furthermore, we use the AWS service *Lambda* for updating the state database

with information from the Kinesis stream. Lambda is a function-as-a-service offer of AWS, which allows running code on-demand, paying for the number of invocations as well as for execution time. The AWS Lambda function for updating the state database is simple. Depending on the state-changing object and the type of state, the function determines the cell address in the database overwrite its value with the state information from Kinesis.

2.4. Execution of the DES model

We also implement the DES model as AWS Lambda function. For online simulation, only those events are relevant to which state variables change. The AWS services DynamoDB and Lambda provide two functions that support an efficient model initialization and execution. Each cell of a DynamoDB table can be configured as a Kinesis stream. In consequence, DynamoDB provides event-based information about cell entry updates. AWS Lambda functions can be defined as stream consumers of DynamoDB cells, allowing the initialization and execution of a DES model in dependence of table updates. For each simulation experiment, all system states are loaded from the state database. The DES model is initialized using the model generation method, which is particularly suitable for online DES models [13; 14]. This method generates flow objects according to the current system state and allocates them to the corresponding processes taking into account a (residual) process time. Finally, the simulation clock is synchronized with the time of the current system state.

2.5. Result persistence and visualization

At the end of a simulation run, the results are written to another DynamoDB table “Result DB”, from where any dashboard application can access and visualize the data.

3. Demonstration of the procedure model on an exemplary problem

For the sake of comprehensibility, we will explain our procedure model using a simple example. The full implementation is accessible on GitHub (link in the abstract of the paper).

3.1. Use case

Fig. 2 shows a fictive and simplified order manufacturing process. Customer orders arrive at random intervals and are first collected (process step "Order Ingest"). In the next step, the materials required for the production of the customer order are requested (process step "Order Material"). The material provision time is defined individually for each customer order. Customer orders wait in the "Wait For Material" process step until the necessary materials are delivered. The "Production" process step can only process one customer order at a time. Customer orders wait in the "Wait For SOP" process step until the downstream process is available for the start of production (SOP).

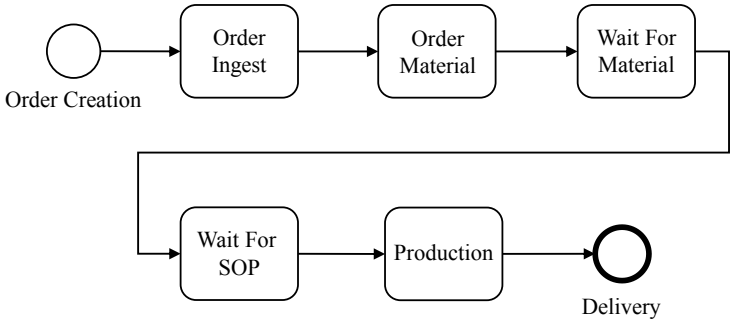

Fig. 2. Exemplary process of an order manufacturer.

3.2. Problem statement

Every time the manufacturer requests material for producing a specific order (process step "Order Material"), an initial estimated time of arrival (ETA) provides information on when the requested material will be available for production. In reality, however, unexpected events in the supply chain may introduce delays at any point in time, which lead to deviations of planned arrival times. Since the production process is a capacity-constrained resource and thus represents a potential bottleneck of the system, any unexpected delay may impede the realization of a predetermined production schedule. Against this background, we want to implement a DES model that can process ETA updates in real time and visualize their impact on the production schedule.

3.3. Implementation

We implement the process of Fig. 2 with the Python package Casymda [15]. Casymda builds up on the DES Python package SimPy [16] and the open-source BPMN modeler Camunda. The main idea of Casymda is using the Camunda modeler as graphical development environment, thus to allow a block-based creation of DES models via drag-on-drop as it is common for many commercial off-the-shelf DES tools. The core of Casymda is a library of reference models (e.g. flow object, resource, source, delay, sink) developed with SimPy3, based on which own DES models can be developed, and a parser that automatically generates DES models based on prebuilt BPMN models. Furthermore, Casymda comes with a simple, yet powerful visualization concept that animates the movement of flow objects along the simulated process and displays state changes of process blocks.

Furthermore, we use Plotly Dash for visualizing results of the simulation. Plotly Dash is a popular framework for analytics web-apps. It allows a quick development of dynamic dashboards using the Python language. Under the hood, Plotly Dash relies on the Python web framework flask and creates React websites with Plotly charts. Our dashboard implementations just contains a simple Gantt chart, which visualizes the production schedule based on current ETA information. In order to allow an automatic refreshment of the dashboard, we implement an interval-callback that polls updates from the database in cyclic manner.

3.4. Setup requirements

The complete cloud-based event-processing pipeline can be easily setup and run locally on the development machine using the tools Docker, Terraform and LocalStack.

Docker is an open-source platform that allows the virtualization and encapsulation of software as standalone packages known as containers. Thus, Docker allows software to run without the need to install additional libraries, packages, and environments to which the software has dependencies. Terraform, on the other hand, is an open-source infrastructure-as-code software tool that automates the configuration and provisioning of cloud services, such as Kinesis streams, DynamoDB tables, and Lambda Functions. For test purposes, we use LocalStack to emulate necessary AWS services locally, i.e. Kinesis, Lambda, and DynamoDB, eliminating the need to pay for actual cloud service usage during development.

To run the complete setup locally, a simple series of steps can be followed (detailed walk-through provided as part of the repository readme):

1. Start LocalStack (e.g., using Docker)
2. Install and use Terraform to provision the needed AWS services
3. Run an emulation of the real system which interacts with the digital twin (e.g., start via Docker, web-based interface to the emulation model)
4. Start the dashboard web-app to visualize the system state (via Docker). A dynamic Gantt chart shows the forecasted effects of supply-chain events (such as created orders and delivery delays).

3.5. Evaluation

Using a web-browser, the dashboard can be accessed, as well as the web-interface of the Casymda emulation model with an animation of the process. Fig. 3 visualizes the run of a simulation experiment with four products.

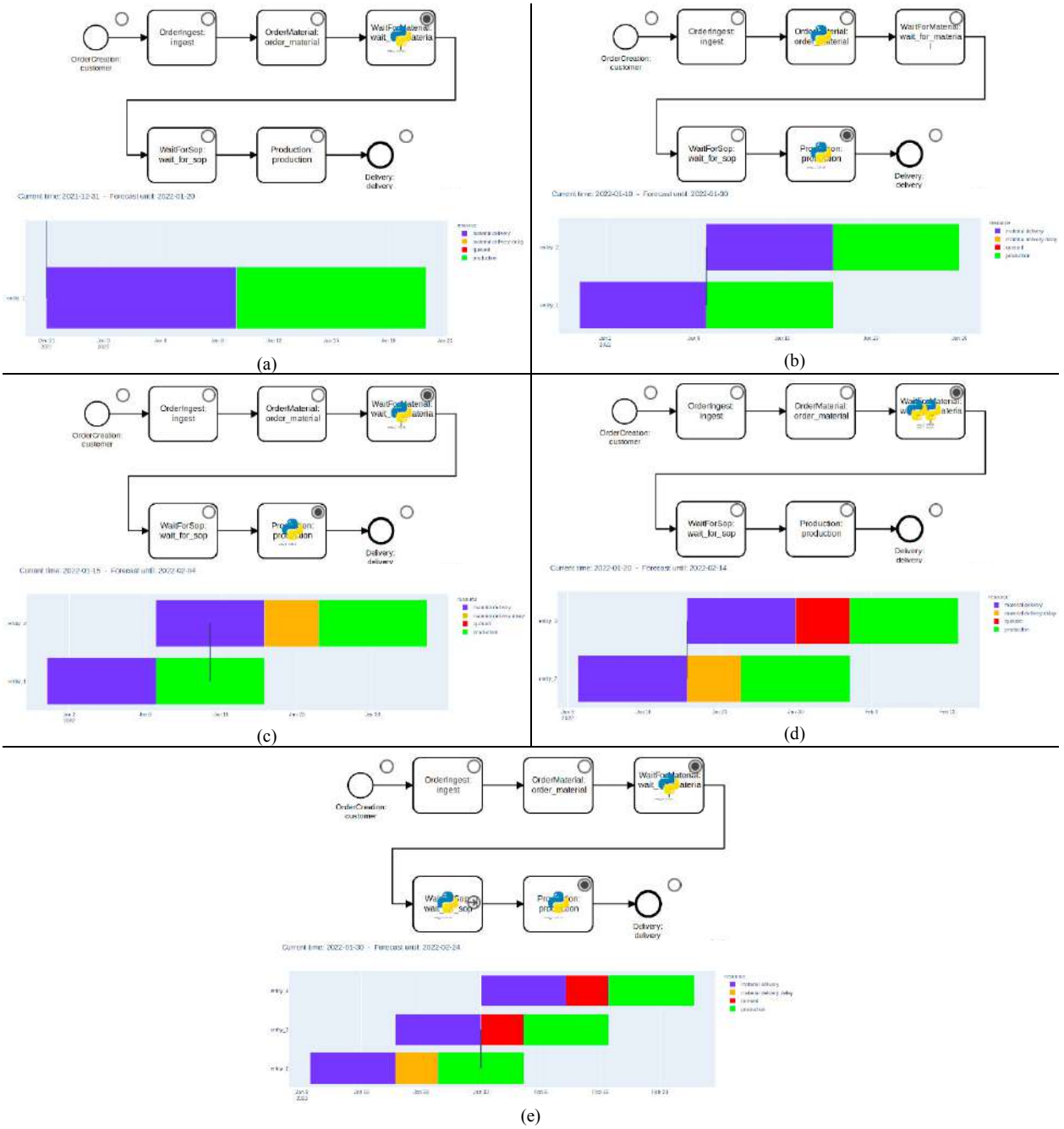

Fig. 3. Simulation run with four products.

When starting the emulation, the model's source will generate orders that flow along the process steps. At the same time, the dashboard updates with a minor delay and visualizes the completion time of all orders that are currently being processed. A vertical line in the chart indicates the point in time when the simulation run started and the forecast was created. At the beginning (a), the source creates the first product. The product waits in the "Wait For Material" process step, until the material for production has arrived. The chart below visualizes the ETA (blue bar) and the duration of the production process including the estimated completion time (green bar). Next (b), the second order arrives in the system, while the first order is in production. The forecast in the Gantt chart below does not show any deviations. However, after the second order waits 5 days for material (c), the DES model receives an information about a delay in the material supply and updates the forecast of the second order (orange bar). The third order arrives (d) and immediately obtains an increase in the estimated completion time. Due to the capacity constraint of the production step (max. one order concurrently), the delay of the second order (orange bar) will also prevent the SOP of the third order on time, even though the material is expected to be available by then (red bar). Finally (e), the source generates the fourth order, which is queued in the "Wait For Material" block. Like for the third order, the model estimates a delay of the completion time of the fourth order, due to the delayed material supply for the second order.

4. Conclusion

In this paper, we presented a procedure model for deploying discrete-event simulation (DES) models on Amazon Web Services (AWS). Furthermore, we demonstrated the application of our procedure model based on a simple use case of an order manufacturing process. The intention of the example was to deliver a minimum proof of concept how the procedure model can be used to guide the deployment and real-time execution of DES models on the AWS cloud. Thus, many extensions are necessary that would complicate the use case and justify the implementation and testing of additional cloud features. In future research, we plan to extend the problem to reach real-life complexity by introducing random process times, material inventories, and different order replenishment strategies into the use case. The resulting problem would particularly benefit from parallelized execution of simulation experiments to determine appropriate inventory management strategies in real time. Although parallelized execution of simulation models is easily possible with the container-based implementation, we currently have not implemented any synchronization strategies that wait for the termination of simulation experiments.

Concerning the analytics of ingested event data, stream processing solutions such as AWS Kinesis Data Analytics might be useful to identify relevant patterns and trigger forecast and optimization runs only in case of critical process deviations. A major gap in our current approach is the fact that we were only able to test our procedure model on an open source and Python-based DES environment. The evaluation and deployment of commercial off-the-shelf DES tools on cloud infrastructures is severely hampered by restrictive licensing and closed-sourced software. While the presented sample implementation is built using AWS-specific services, similar compute and storage services are available on different cloud platforms, so that a generalization of the approach should also be possible.

References

[1] Fujimoto, R. M. Parallel and distributed simulation systems; Wiley: New York (USA), 2000.

[2] Fujimoto, R. M. Parallel discrete event simulation. Commun. ACM 33, 10; pp. 30–53.

[3] Malik, A.; Park, A.; Fujimoto, R. Optimistic Synchronization of Parallel Simulations in Cloud Computing Environments. In Proceedings of the 2009 IEEE International Conference on Cloud Computing (CLOUD), Bangalore, Indien, 21.-25. September 2009 IEEE: Piscataway, NJ, 2009; pp. 49–56.

[4] Yoginath, S. B.; Perumalla, K. S. Efficient Parallel Discrete Event Simulation on Cloud/Virtual Machine Platforms. ACM Trans. Model. Comput. Simul. 26, 1; pp. 1–26.

[5] Jafer, S.; Liu, Q.; Wainer, G. Synchronization Methods in Parallel and Distributed Discrete-Event Simulation. Simulation Modelling Practice and Theory 30; pp. 54–73.

[6] Vanmechelen, K.; Munck, S. de; Broeckhove, J. Conservative Distributed Discrete Event Simulation on Amazon EC2. In Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Ottawa (Kanada), 13.-16. Mai 2012; Balaji, P., Ed.; IEEE: Piscataway, NJ, 2012; pp. 853–860.

[7] Liu, X.; He, Q.; Qiu, X.; Chen, B.; Huang, K. Cloud-based Computer Simulation. Towards Planting Existing Simulation Software into the Cloud. Simulation Modelling Practice and Theory 26; pp. 135–150.

[8] Heavey, C.; Dagkakis, G.; Barlas, P.; Papagiannopoulos, I.; Robin, S.; Mariani, M.; Perrin, J. Development of an Open-Source Discrete Event Simulation Cloud Enabled Platform. In Proceedings of the 2014 Winter Simulation Conference (WSC), Savanah (USA), 07.-10. Dezember 2014; Tolk, A., Ed.; IEEE: Piscataway, NJ, 2014; pp. 2824–2835.
[9] Padilla, J. J.; Diallo, S. Y.; Barraco, A.; Lynch, C. J.; Kavak, H. Cloud-based Simulators. Making Simulations Accessible to Non-Experts and Experts Alike. In Proceedings of the 2014 Winter Simulation Conference (WSC), Savanah (USA), 07.-10. Dezember 2014; Tolk, A., Ed.; IEEE: Piscataway, NJ, 2014; pp. 3630–3639.
[10] Borshchev, A.; Churkov, N. Anylogic Cloud. Cloud-Based Simulation Analytics. In Proceedings of the 2018 Winter Simulation Conference (WSC), Göteborg (Schweden), 09.-12. Dezember 2018; Johansson, B., Jain, S., Eds.; IEEE: Piscataway, NJ, 2018; pp. 4245.
[11] Amazon Leads $100 Billion Cloud Market. Available online: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ (accessed on 27.01.2021).
[12] Donhauser, T., 2020 Ressourcenorientierte Auftragsregelung in einer hybriden Produktion mittels betriebsbegleitender Simulation. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik).
[13] Hanisch, A.; Tolujew, J.; Schulze, T. Initialization of Online Simulation Models. In Proceedings of the 2005 Winter Simulation Conference (WSC), Orlando (USA), 04.-07. Dezember; Kuhl, M. E., Ed.; IEEE: Piscataway, NJ, 2005; pp. 1795–1803.
[14] Hotz, I., 2007 Simulationsbasierte Frühwarnsysteme zur Unterstützung der operativen Produktionssteuerung und -planung in der Automobilindustrie. Dissertation. Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik).
[15] Hofmann, W., 2020 Casymda. Camuda-Modeler-based creation of SimPy discrete event simulation models. Available online: https://github.com/fladdimir/casymda (accessed on 29.05.2021).
[16] Matloff, N., 2008 Introduction to Discrete-Event Simulation and the SimPy Language. Available online: http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf (accessed on 24.11.2020).