Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

SIM4EDU.COM – WEB-BASED SIMULATION FOR EDUCATION

Gerd Wagner

Department of Informatics
Brandenburg University of Technology
P. O. Box 101344
03013 Cottbus, GERMANY

ABSTRACT

The sim4edu.com project website supports web-based simulation with open source technologies for open science and education. It provides both simulation technologies and a library of educational simulation examples. Its aim is to support both the use and the development of various kinds of simulations, including *ad-hoc* simulations, *Cellular Automata* models, *NetLogo*-style grid space models, *discrete event* simulation and *agent*-based simulation. Sim4edu facilitates building state-of-the-art user interfaces for web-based simulations and simulation games without requiring simulation developers to learn all the recent web technologies involved (e.g., HTML5, CSS3, SVG and WebGL). Using the power of the web, Sim4edu allows researchers and educators to publish and share their models easily.

1 INTRODUCTION

The *Simulation for Education (Sim4edu)* project website supports web-based simulation with open source technologies for science and education. It provides both simulation technologies and educational simulation examples. Sim4edu facilitates building state-of-the-art user interfaces for simulations and simulation games without requiring simulation developers to learn all the recent web technologies involved (e.g., HTML5, CSS3, SVG and WebGL).

As opposed to traditional simulation technologies, web-based simulations, typically implemented with JavaScript, can be executed in any web browser, not just on desktop computers, but also on mobile devices like tablets and smartphones. This allows sharing simulations by means of simple web links and makes them easily accessible to anyone anywhere.

Wagner (2012) has argued that, for a number of reasons, but mainly due to a lack of open educational resources, simulation is not widely used in education today. Being open means both being legally as well as financially accessible and being technically/effectively accessible to all learners, which implies being web-based with special consideration of the mobile Web. Sim4edu aims at supporting the creation, publication and reuse of simulation models as *Open Educational Resources*.

The first simulator provided by Sim4edu, *Omega-Epsilon* (ΩE), is a JavaScript implementation of *Object-Event Simulation (OES)*, which is a successor project to the one reported in (Wagner 2012). It supports both *next-event time progression*, as used in discrete event simulation, and *fixed-increment time progression*, as used in *NetLogo*-based social science simulations as well as in continuous state change simulations. The next simulator on the roadmap of Sim4edu is called *Alpha-Omega-Epsilon* (AΩE), which implements *Agent/Object-Event Simulation (A/OES)* supporting agent-based discrete event simulation.

For any simulator based on a certain simulation language, which is itself based on a certain simulation paradigm, there is the usability issue of coding-based versus user-interface-based simulation development. The more basic approach is coding: using either a specific simulation programming language or using a general-purpose programming language with a library that implements the simulation language.

This development approach requires coding skills and may not be an option for those subject matter experts who are interested in simulation, but are not willing to learn coding.

The alternative to coding is using a simulation development tool with a set of user interfaces (UI) allowing to develop a simulation by a combination of

- entering values into form fields
- choosing values/options from selection lists (or other choice widgets such as radio/checkbox groups or date pickers)
- making a model diagram

It is obvious that in terms of usability, a UI-based simulation development environment is preferable. This was also the selling point and reason for the big success of *Arena* (Pegden and Davis 1992). All commercially successful simulation software packages support the UI-based approach today.

The Sim4edu platform does currently require JavaScript coding, and does not offer a UI-based simulation development environment. But there are plans to provide such a tool in the future.

2 RUNNING SIM4EDU SIMULATIONS USING DYNAMIC ON-DEMAND DISTRIBUTION

JavaScript allows running the same code either in a web browser on a front-end device, such as a mobile phone, or on a backend (cloud) machine with the execution environment *NodeJS*. The decision where to run a simulation program can be taken at runtime, possibly depending on available computing resources.

We can illustrate this powerful dynamic distribution approach with an example. The JavaScript implementation of the service desk simulation discussed in Section 3 and 4 can be run in two forms:

1. On the Heroku cloud platform using the hyperlink https://oesimulator.herokuapp.com/sims/1
2. In your web browser using the hyperlink https://oesimulator.herokuapp.com/sims/1?local=true

3 INTRODUCTION TO OBJECT-EVENT SIMULATION

We illustrate the main concepts of OES with an example. We model a system of one or more service desks, each of them having its own queue, as a discrete event system:

1. Customers arrive at a service desk at random times.
2. If there is no other customer in front of them, and the service desk is available, they are served immediately, otherwise they have to queue up in a waiting line.
3. The duration of services varies, depending on the individual case.
4. When a service is completed, the customer departs and the next customer is served, if there is still any customer in the queue.

3.1 Making a Conceptual Model

We only make a simplified conceptual model consisting of two sets of potentially relevant object types and event types extracted from the system description. The potentially relevant **object types** of the problem domain are:

- Customer,
- ServiceDesk,
- WaitingLine,
- ServiceClerk, if the service is performed by (one or more) clerks.

The potentially relevant **event types** are:

- CustomerArrival,
- StartOfService,
- EndOfService,
- CustomerDeparture.

3.2 Making an Information Design Model

When making a simulation model, the right degree of abstraction depends on the purpose of the model. But abstracting away from too many things may make a model too unnatural and not sufficiently generic, implying that it cannot be easily extended to model additional features (such as more than one service desk).

In the case of our example, if the purpose of the simulation model is to compute the service utilization and the maximum queue length, only, then we may abstract away from the following object types:

- Customer: we don't need any information about individual customers.
- WaitingLine: we don't need to know who is next, it's sufficient to know the length of the queue.
- ServiceClerk: we don't need any information about the service clerk(s).

Notice that, for simplicity, we consider the customer that is currently being served to be part of the queue. In this way, in the simulation program, we can check if the service desk is busy by testing if the length of the queue is greater than 0. Consequently, we can model the system state in terms of (one or more) ServiceDesk objects having only one property: queueLength (a non-negative integer).

We also look for opportunities to simplify our event model by dropping event types that are not needed, e.g., because their events coincide with events of another type. This is the case with EndOfService events and CustomerDeparture. Consequently, we can drop the event type EndOfService.

There are two situations when a new service can be started: either when the waiting line is empty and a new customer arrives, or when the waiting line is not empty and a service ends. Therefore, any StartOfService event immediately follows either a CustomerArrival or a CustomerDeparture event, and we may abstract away from the StartOfService event and drop it from the model.

So we only need to model two event types: CustomerArrival and CustomerDeparture. The event type CustomerArrival is an example of a type of ***exogenous*** events, which are not caused by any causal regularity of the system under investigation and, therefore, have to be modeled with a (typically stochastic) ***recurrence*** function that allows to compute the time of the next occurrences of events of that type. In OES, exogenous event types are a built-in concept such that an OES simulator takes care of creating the next exogenous event whenever an event of that type is processed. This mechanism makes sure that there is a continuous stream of exogenous events throughout a simulation run.

We also have to model the random variations of two variables: the time in-between two customer arrival events and the service duration. We model the time in-between two customer arrival events as a discrete random variable with a uniform distribution between 1 and 6 minutes, symbolically U(1,6). We model the service duration random variable with an empirical distribution of 2 with probability 0.3, 3 with probability 0.5 and 4 with probability 0.2.

Computationally, object types and event types correspond to classes, either of an object-oriented information model, such as a UML class diagram, or of a computer program written in an object-oriented programming language, such as Java or JavaScript. In our example, the object class ServiceDesk has a property queueLength, and the event classes CustomerArrival and CustomerDeparture have a property serviceDesk referencing the service desk at which an event occurs.

In addition, in a class model, random variables like the service duration, can be expressed as class-level ("static") methods in the class to which they belong. Thus, we get the class diagram shown in Fig. 1.

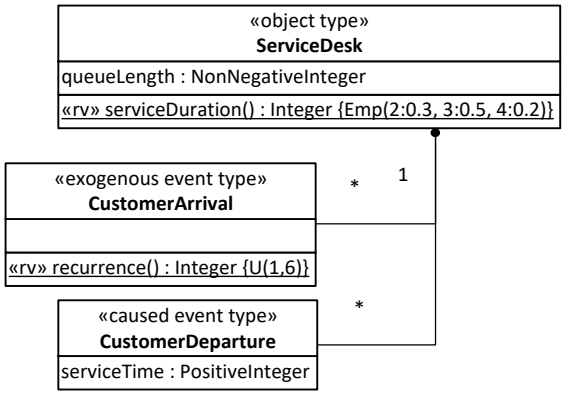

Figure 1: An information design model.

Notice that both event types, *CustomerArrival* and *CustomerDeparture*, have a many-to-one association with the object type *ServiceDesk*. This expresses the fact that any such event occurs at a service desk, which participates in the event. This association is implemented in the form of a reference property serviceDesk in each of the two event types.

In addition to this information model, we need to make a process model, which captures the dynamics of the service desk system consisting of arrival and departure events triggering state changes and follow-up events.

3.3 Making a Process Design Model

A process model can be expressed with the help of event rules, which define what happens when an event (of a certain type) occurs, or, more specifically, which state changes and which follow-up events are caused by an event of that type.

Event rules can be expressed with the help of pseudo-code or in process diagrams, or in a simulation or programming language. Table 1 shows the two event rules defining the transition logic of a service desk system, expressed in pseudo-code.

Table 1: Event rule table specifying rules in pseudo-code.

ON (event type)	**DO (event routine)**

CustomerArrival(sd) @ t
 	
with sd : ServiceDesk
	`INCREMENT sd.queueLength`
	`IF sd.queueLength = 1`
	`THEN SCHEDULE CustomerDeparture( sd) @`
	`  (t + ServiceDesk.serviceDuration())`

CustomerDeparture(sd) @ t
 	
with sd : ServiceDesk
	`DECREMENT sd.queueLength`
	`IF sd.queueLength > 0`
	`THEN SCHEDULE CustomerDeparture( sd) @`
	`  (t + ServiceDesk.serviceDuration())`

In the next section, we discuss how to implement this simple model of a service desk system with the Sim4edu framework.

4 MAKING SIMULATIONS WITH SIM4EDU OMEGA-EPSILON

The Sim4edu Omega-Epsilon (ΩE) simulator implements the Object-Event Simulation Language OESL, which represents a general Discrete Event simulation approach based on the object-event worldview. In OESL, a model normally defines various types of objects and events, but OESL also supports

1. models without objects, if they define state variables in the form of global variables, instead;
2. models without events, if they use pure fixed-increment time progression (by defining an `OnEachTimeStep` procedure and a `timeIncrement` parameter), instead; such a model can be used
    a. as a discrete model that abstracts away from explicit events and uses only implicit time events ("ticks"), which is a popular approach in social science simulation, or
    b. for modeling continuous state changes (e.g. objects moving in a continuous space).

Using a simulation framework like Sim4edu ΩE means that the model-specific logic has to be coded in the form of object types, event types, event routines and other functions for model-specific computations, but not the general simulator operations (e.g. time progression and statistics) and the environment handling (e.g. user interfaces for statistics output and visualization).

The following sections discuss the basic concepts of OESL and the Sim4edu ΩE simulator, and show how to implement the simple service desk model described in the previous section. It is possible to run this simulation model and download the code from the Sim4edu website.

4.1 Simulation Time

A simulation model has an underlying **time model**, which can be either the abstract model of discrete time, when setting

```
sim.model.time = "discrete";
```

or the concrete model of continuous time, when setting

```
sim.model.time = "continuous";
```

Choosing a discrete time model means that time is measured in steps (with equal durations), and all random time variables used in the model need to be discrete (i.e., based on discrete probability distributions). Choosing a continuous time model means that one has to define a simulation time granularity, as explained in the next sub-section.

In both cases, the underlying simulation **time unit** can be either left unspecified (in the case of an abstract time model), or it can be set to one of the time units "ms", "s", "m", "h", "D", "W", "M" or "Y", as in

```
sim.model.timeUnit = "h";
```

4.1.1 Time Granularity

When a simulation model is based on continuous time, it is possible to control the time granularity (the time delay until the next moment) in one of two ways:

1. through simulation time rounding by setting the model parameter timeRoundingDecimalPlaces to a suitable value, which implies a corresponding value of the model parameter nextMomentDeltaT;
2. by explicitly setting the model parameter nextMomentDeltaT.

The model parameter nextMomentDeltaT is used by the simulator for scheduling next events with a minimal delay.

4.1.2 Time Progression

An important issue in simulation is the question how the simulation time is advanced by the simulator. The OES paradigm supports **fixed-increment** time progression and **next-event** time progression, and their combination.

A Sim4edu ΩE model with pure fixed-increment time progression defines an `OnEachTimeStep` procedure and a `timeIncrement` parameter, but no event types. Such a model can be used

1. for modeling continuous state changes (e.g. objects moving in a continuous space), or
2. as a discrete model that abstracts away from explicit events and uses only implicit periodic time events ("ticks"), which is a popular approach in social science simulation.

A simulation model with pure next-event time progression, representing a classical DES model, defines event types and event rules, but no timeIncrement parameter.

It is also possible to combine both time progression mechanisms, e.g., in a "hybrid" model that supports both discrete and continuous state changes, or in a social science model based on "ticks" and explicit events.

4.1.3 Real-Time Simulation

Real-time simulation means to run an observable simulation model in such a way that the speed of its state changes is close to the speed of the state changes in the simulated real-world system. This is only possible if the simulator is able to run the simulation at least as fast as the real-world system is running. If this is the case, the running simulator can be slowed down to real-time speed.

In the case of fixed-increment time progression with a timeUnit and real-time simulation turned on (by setting the scenario parameter realtimeFactor to 1), the simulator delays each simulation step such that its real duration is equal to its simulation time, which is timeIncrement [timeUnit].

In the case of fixed-increment time progression without a timeUnit (that is, with abstract time), the simulator cannot automatically run in real-time, but the scenario parameter stepDuration (for specifying the real duration of a simulation step) can be set to a suitable value for making the simulation observable in real-time.

4.1.4 Simulation Models

A simulation model essentially defines the state structure and the dynamics of the simulated system. While the system's state structure is defined by the types of objects that populate it, its dynamics is defined by certain types of events and the state changes and follow-up events caused by them. We model the state structure of a simulated system with the help of global variables and object types, and we model its dynamics with the help of event types and event rules, such that, for any event type, an event rule specifies the state changes of affected objects and the follow-up events caused by the occurrence of an event of that type.

A **simulation model** essentially consists of:

1. a choice of time model (either discrete time or continuous time);
2. possibly a choice of space model, if the simulation is about objects in space;
3. a set of object type and event type definitions (and possibly other entity types allowing to model activities, agents, actions, etc.);

4. a set of event rules, which capture causal regularities governing the system's state changes and the causation of follow-up events.

We now show how to implement the object and event types defined by the information design model shown in Fig. 1.

4.1.5 Object Types

Object types are defined in the form of classes (more precisely, as instances of the meta-class cLASS). As an example, we define an object type for service desks with the attribute `queueLength`:

```
var ServiceDesk = new cLASS({
  Name: "ServiceDesk",
  supertypeName: "oBJECT",
  properties: {
    "queueLength": { range: "NonNegativeInteger",
        initialValue: 0, label: "Queue length"}
  }
});
```

Notice that, in Sim4edu ΩE, object types are defined as subtypes of the pre-defined class `oBJECT`, from which they inherit an integer-valued `id` attribute. The discrete random variable for modeling random service durations, which samples integers between 2 and 4 from an empirical probability distribution, is implemented as a class-level function `serviceDuration` in the ServiceDesk class:

```
ServiceDesk.serviceDuration = function () {
  var r = rand.uniformInt( 0, 99);
  if ( r < 30) return 2;          // probability 0.30
  else if ( r < 80) return 3;     // probability 0.50
  else return 4;                  // probability 0.20
};
```

4.1.6 Event Types

We distinguish between two kinds of events:

1. caused events are caused by other events occurring during a simulation run;
2. exogenous events seem to happen spontaneously, but may be caused by factors, which are external to the simulation model.

Here is an example of an exogenous event type definition:

```
var CustomerArrival = new cLASS({
  Name: "CustomerArrival",
  supertypeName: "eVENT",
  properties: {
    "serviceDesk": {range: "ServiceDesk"}
  },
  methods: {
    "onEvent": function () {
      ...
    }
  }
});
```

Notice that the event type CustomerArrival includes a reference property `serviceDesk`, which is used for referencing the service desk object at which an event occurs. Each event type needs to define an `onEvent` method, which implements the event rule for events of the defined type. Event rules are discussed below.

Typically, exogenous events occur periodically. They are therefore defined with a recurrence function, which provides the time in-between two events (often in the form of a random variable), and with a createNextEvent function, which invokes the recurrence function. The recurrence function and a createNextEvent function (with a parameter e for the current event) are defined as class-level methods:

```
CustomerArrival.recurrence = function () {
  return rand.uniformInt( 1, 6);
};
CustomerArrival.createNextEvent = function (e) {
 return new CustomerArrival({
     occTime: e.occTime + CustomerArrival.recurrence(),
     serviceDesk: e.serviceDesk
  });
};
```

Notice that the recurrence of CustomerArrival events is defined in terms of a discrete random variable based on the uniform distribution providing integers between 1 and 6. The createNextEvent function is invoked by the simulator for creating, and scheduling, the next event whenever an event of the given exogenous event type occurs.

In our example model of a service desk system, any customer departure event is caused, either by a customer arrival event or by a preceding service start event.

```
var CustomerDeparture = new cLASS({
  Name: "CustomerDeparture",
  supertypeName: "eVENT",
  properties: {
    "serviceTime": {range: "NonNegativeInteger"},
    "serviceDesk": {range: "ServiceDesk"}
  },
  methods: {
    "onEvent": function () {
      ...
  }
});
```

4.1.7 Event Rules

An event rule for an event type defines what happens when an event of that type occurs, by specifying the caused state changes and follow-up events. In Sim4edu ΩE, an event rule for an event type is defined as a method `onEvent` of the class that implements the event type. This method, which is also called event routine, returns a set of events (more precisely, JS objects representing events).

The following event rule method is defined in the `CustomerArrival` class.

```
// CustomerArrival event rule
"onEvent": function () {
  var srvTm=0, changes = [], events = [];
  this.serviceDesk.queueLength++;
  sim.stat.arrivedCustomers++;
  // if the service desk is not busy
  if (this.serviceDesk.queueLength === 1) {
```

```
    srvTm = ServiceDesk.serviceDuration();
    events.push( new CustomerDeparture({
      occTime: this.occTime + srvTm,
      serviceTime: srvTm,
      serviceDesk: this.serviceDesk
    }));
  }
  return events;
}
```

The context of this event rule method is the event that triggers the rule, that is, the variable `this` references a JS object that represents the triggering event. Thus, the expression `this.serviceDesk` refers to the service desk object associated with the current customer arrival event, and the statement `this.serviceDesk.queueLength++` increments the queueLength attribute of this service desk object (as an immediate state change).

The following event rule method is defined in the `CustomerDeparture` class.

```
// CustomerDeparture event rule
"onEvent": function () {
  var changes = [], events = [], srvTm=0;
  // remove customer from queue
  this.serviceDesk.queueLength--;
  // if there are still customers waiting
  if (this.serviceDesk.queueLength > 0) {
    // start next service and schedule its end/departure
    srvTm = ServiceDesk.serviceDuration();
    events.push( new CustomerDeparture({
      occTime: this.occTime + srvTm,
      serviceTime: srvTm,
      serviceDesk: this.serviceDesk
    }));
  }
  sim.stat.departedCustomers++;
  sim.stat.totalServiceTime += this.serviceTime;
  return events;
}
```

4.2 Simulation Scenarios

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, or with different visualizations.

A **simulation scenario** consists of

1. simulation parameter settings, such as setting a value for simulationEndTime,
2. a simulation model,
3. an initial state definition, and
4. optional user interface (UI) definitions of, e.g., a statistics UI and an observation (or visualization) UI.

An empty template for a simulation scenario has the following structure:

```
// ***** Simulation Parameters **************
sim.scenario.simulationEndTime = ...;
```

```
sim.scenario.randomSeed = ...;  // optional
sim.scenario.createLog = ...;  // true/false
sim.scenario.visualize = ...;  // true/false
// ***** Simulation Model ******************
sim.model.name = "...";
sim.model.time = "...";  // discrete or continuous
sim.model.objectTypes = [...];
sim.model.eventTypes = [...];
// ***** Initial State **********************
sim.scenario.initialState.objects = {...};
sim.scenario.initialState.events = {...};
// Ex-Post Statistics
sim.model.statistics = {...};
```

We briefly discuss each group of scenario information items in the following sub-sections.

4.2.1 Simulation Parameters

Simulation parameters are defined as attributes of the simulation scenario. The most important parameters are:

- `simulationEndTime`: this mandatory attribute defines the duration of a simulation run;
- `stepDuration`: an optional attribute for specifying a minimum execution-time duration (in milliseconds) for each simulation step. This can be used for delaying simulation steps such that the simulation runs in real-time.
- `randomSeed`: Setting this optional parameter to a positive integer allows to obtain a specific fixed random number sequence generated by the random number generator. This can be useful for being able to test a simulation model by testing if expected results are obtained.
- `visualize`: A Boolean parameter that allows to turn on/off any visualization defined by the scenario.
- `createLog`: A Boolean parameter that allows to turn on/off the simulation log.

4.2.2 Initial State

Defining an initial state means:

1. assigning initial values to global variables, if there are any;
2. defining which objects exist initially, and assigning initial values to their properties;
3. defining which events are scheduled initially.

A scenario must include an initial state definition, which consists of a set of object definitions and a set of initial event definitions. An initial state object is defined as an entry in the map initialState.objects such that the object's ID is the map entry's key, and the map entry's value is a set of property-value slots, including a slot for the special attribute `typeName` defining the object's type, as shown in the following example:

```
sim.scenario.initialState.objects = {
  "1": {typeName: "ServiceDesk", name:"serviceDesk1", queueLength: 0}
};
```

Notice that object IDs are positive integers.

An initial event is defined as an element of the array list `initialState.events` in the form of a set of property-value slots, including a slot for the special attribute `typeName` defining the event's type, as shown in the following example:

```
sim.scenario.initialState.events = [
  {typeName: "CustomerArrival", occTime:1, serviceDesk:"1"}
];
```

4.3 Statistics

In scientific and engineering simulation projects the main goal is getting estimates of the values of certain variables or performance indicators with the help of statistical methods. In educational simulations, statistics can be used for observing simulation runs and for learning the dynamics of a simulation model.

For collecting statistics, suitable statistic variables have to be defined. The following code defines statistics variables for the service desk model.

```
sim.model.statistics = {
  "arrivedCustomers": {range:"NonNegativeInteger", label:"Arrived customers"},
  "departedCustomers": {range:"NonNegativeInteger", label:"Departed customers"},
  "totalServiceTime": {range:"NonNegativeInteger"},
  "serviceUtilization": {range:"Decimal", label:"Service utilization",
      computeOnlyAtEnd: true, decimalPlaces: 1, unit: "%",
      expression: function () {
        return sim.stat.totalServiceTime / sim.time * 100
      }
  },
  "maxQueueLength": {objectType:"ServiceDesk", objectIdRef: 1,
      property:"queueLength", aggregationFunction:"max",
      label:"Max. queue length"},
  "averageQueueLength": {objectType:"ServiceDesk", objectIdRef: 1,
      property:"queueLength", aggregationFunction:"avg",
      label:"Avg. queue length"},
  "queueLength": {objectType:"ServiceDesk", objectIdRef: 1,
      property:"queueLength", showTimeSeries: true, label:"Queue length"}
};
```

The first three statistics variables (arrivedCustomers, departedCustomers and totalServiceTime) are simple variables that are updated in event rule methods.

The `serviceUtilization` variable is only computed at the end of a simulation run by evaluating the expression specified for it (dividing the total service time by the simulation time). In the case of the remaining three variables, the data source is the object property queueLength of the service desk object with id=1. For the variable `maxQueueLength` the built-in aggregation function max is applied to this data source, computing the maximum of all queueLength values, while for the variable `averageQueueLength` the aggregation function `avg` is applied. The last variable, `queueLength`, is defined for the purpose of getting a time series chart.

The statistics results are shown in a default view of the statistics output. It is an option to define a non-standard user interface for the statistics output.

5 RELATED WORK

Fishwick (1996) makes an attempt to assess the effects of the wide availability of the early Web on the use of simulation. He points out the new opportunities for re-using simulation resources.

In (Padilla et al 2014), the cloud-based simulation platform ClouDES is presented. It provides a JavaScript-based visual modeling user interface for defining processing network models that are stored in the cloud and executed by running the Java simulator DESMO-J. While Sim4edu supports various

simulation paradigms, including grid space models and processing network models, ClouDES only supports processing network models. As opposed to Sim4edu JavaScript simulations, ClouDES simulations may not scale to a large number of users without an expensive load balancing approach using multiple backend servers, as all simulations run on backend servers.

Fortmann-Roe (2014) presents a powerful JavaScript simulation platform called "Insight Maker", which supports drag & drop model editing in a browser and has a larger user base. While it is technically more mature than Sim4edu, and supports both System Dynamics models and a form of agent-based models, it is not based on a simulation language like OESL and does not support basic DES and processing network models.

6 CONCLUSIONS

The Sim4edu project aims at promoting the use of simulation in science and education by allowing to publish and share all kinds of simulations easily on the Web with the help of JavaScript-based simulation programs. It also aims in educating the science and education communities in the use of modern web technologies for simulation. A further ambitious goal is the provision of web-based environments that allow developing simulations without much coding with the help of suitable user interfaces.

REFERENCES

Fishwick, P. 1996. Web-based Simulation: Some Personal Observations. In *Proceedings of the 1996 Winter Simulation Conference*, edited by J.M Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 772−779.

Fortmann-Roe, S. 2014. Insight Maker: A General-Purpose Tool for Web-Based Modeling & Simulation. *Simulation Modelling Practice and Theory* 47: 28−45.

Padilla, J.J., S.Y. Diallo, A. Barraco, H. Kavak and C.J. Lynch. 2014. "Cloud-Based Simulators: Making Simulations Accessible to Non-Experts and Experts Alike". In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk et al. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 3630–3639.

Pegden, C.D. and D.A. Davis. 1992. Arena: a SIMAN/Cinema-Based Hierarchical Modeling System. In *Proceedings of the 1992 Winter Simulation Conference*, edited by R.C. Crain, J.R. Wilson, J.J. Swain, and D. Goldsman. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 390–399.

Wagner, G. 2012. Simurena − A Web Portal for Open Educational Simulation. In *Proceedings of the 2012 Winter Simulation Conference*, edited by J. Laroque, Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 1588−1599.

AUTHOR BIOGRAPHIES

GERD WAGNER is Professor of Internet Technology in the Dept. of Informatics, Brandenburg University of Technology, Germany, and Adjunct Associate Professor in the Dep. of Modeling, Simulation and Visualization Engineering, Old Dominion University, Norfolk, VA, USA. His research interests include modeling and simulation, foundational ontologies, knowledge representation and web engineering. His email address is G.Wagner@b-tu.de.