Brandenburgische
Technische Universität
Cottbus - Senftenberg

Introduction to JavaScript-based Simulation

Luis Gustavo Nardin

Summer Simulation Conference 2019

Berlin, 22 July 2019

Outline

- Web-Based Simulation (WBS)
  - Architectures
  - Enabling Technologies
  - General-Purpose Platforms
  - Performance
  - Advantages and Disadvantages
- JavaScript-Based Simulation with OESjs
  - Introduction to JavaScript
  - Purchasing Model Description
  - Purchasing Model Implementations

Web-based Simulation

- Broadly defined as the integration of **web technologies** with the **field of simulation**
- **Web resources and technologies** are used to implement the interaction between
  - **simulation engines** and **visualization components**
  - **simulation tools** and **users**
- The **web browser**, in particular, **plays a central role** in
  - the interaction with users
  - performing other simulation tasks depending on the architecture adopted

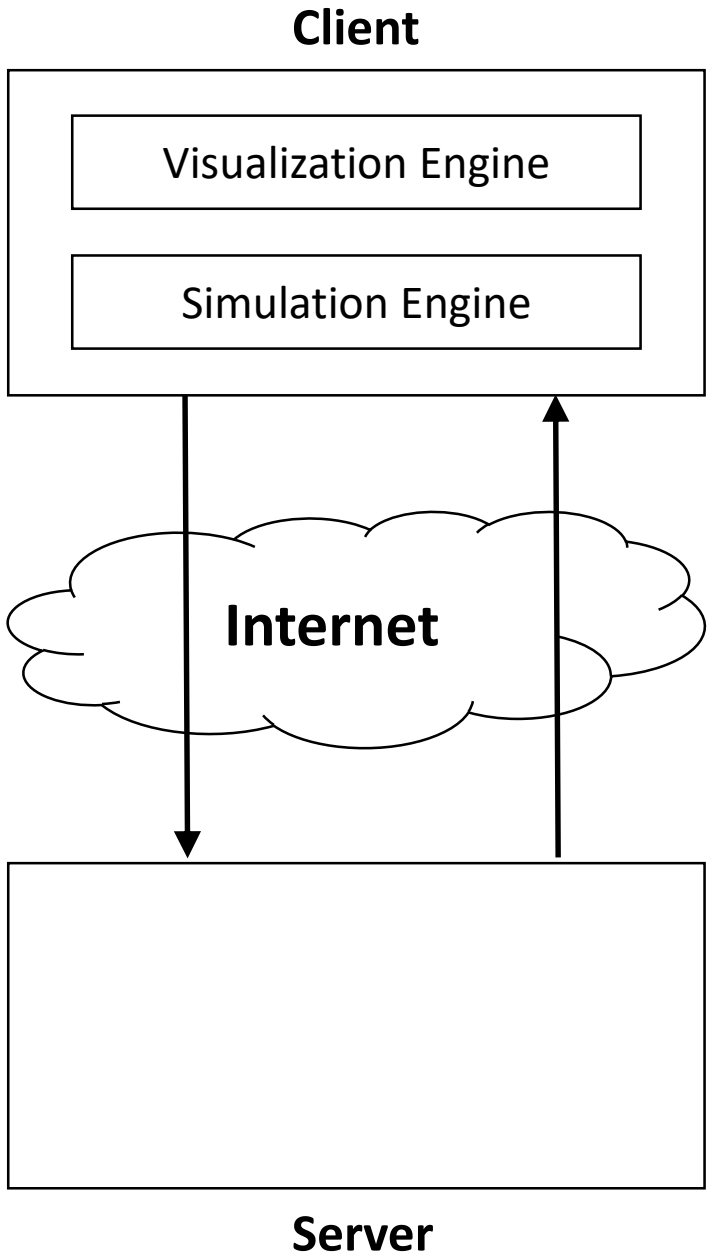
Architectures

There are three main web-based simulation architectures depending where the simulation engine and visualization components are executed:

- **Local**: both components are downloaded to and executed locally in the client computer's web browser
- **Remote**: both components are executed in the server, while the web browser in the client computer works as an interface for submitting simulation jobs and displaying their results
- **Hybrid**: the simulation engine is executed in the server, while the visualization components are executed in the client computer's web browser

Local Architecture

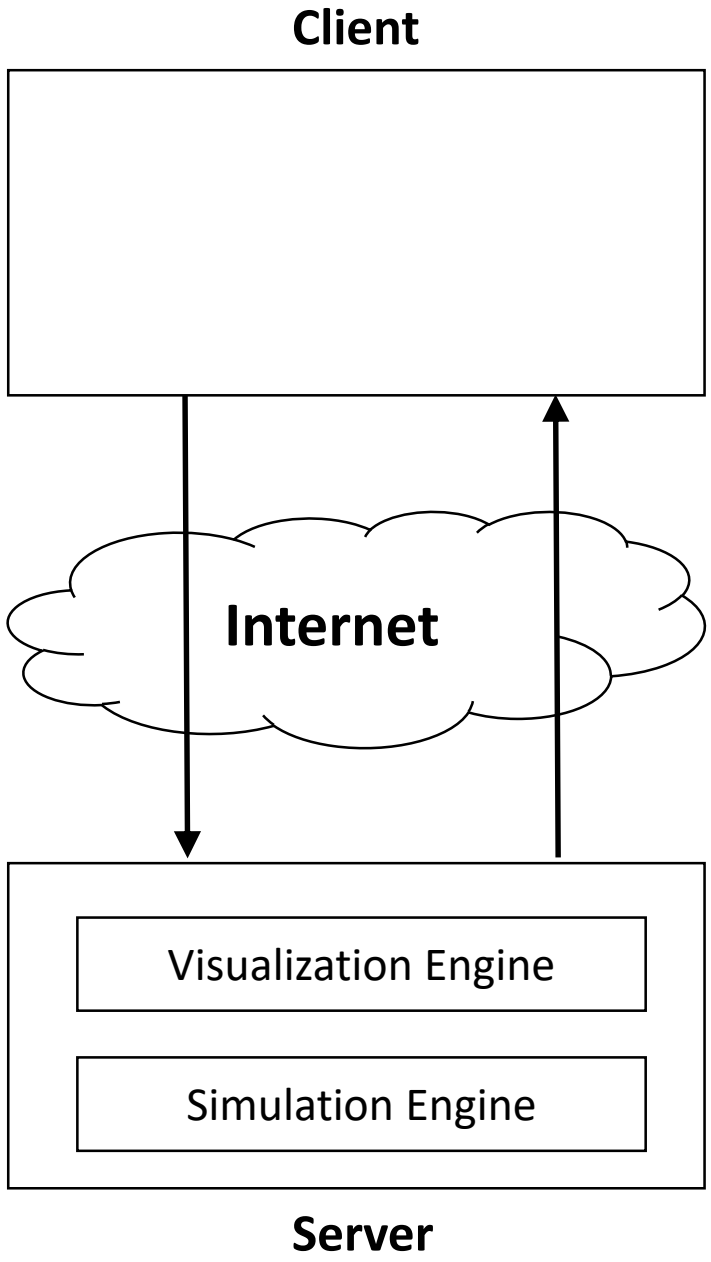
- Require **no communication** between the **client** and **server** after the initial loading phase
- The **simulation execution relies entirely on the client** computer's processing power and there is **no dependency on the communication infrastructure**
- Simulation server is a central distribution point to the simulation, but it does not perform any real simulation task
- **Advantage**: Network latency between the user and the simulator is non-existent
- **Disadvantage**: Simulation execution depends completely on the power of the client machine



(Byrne et al. 2010)

Remote Architecture

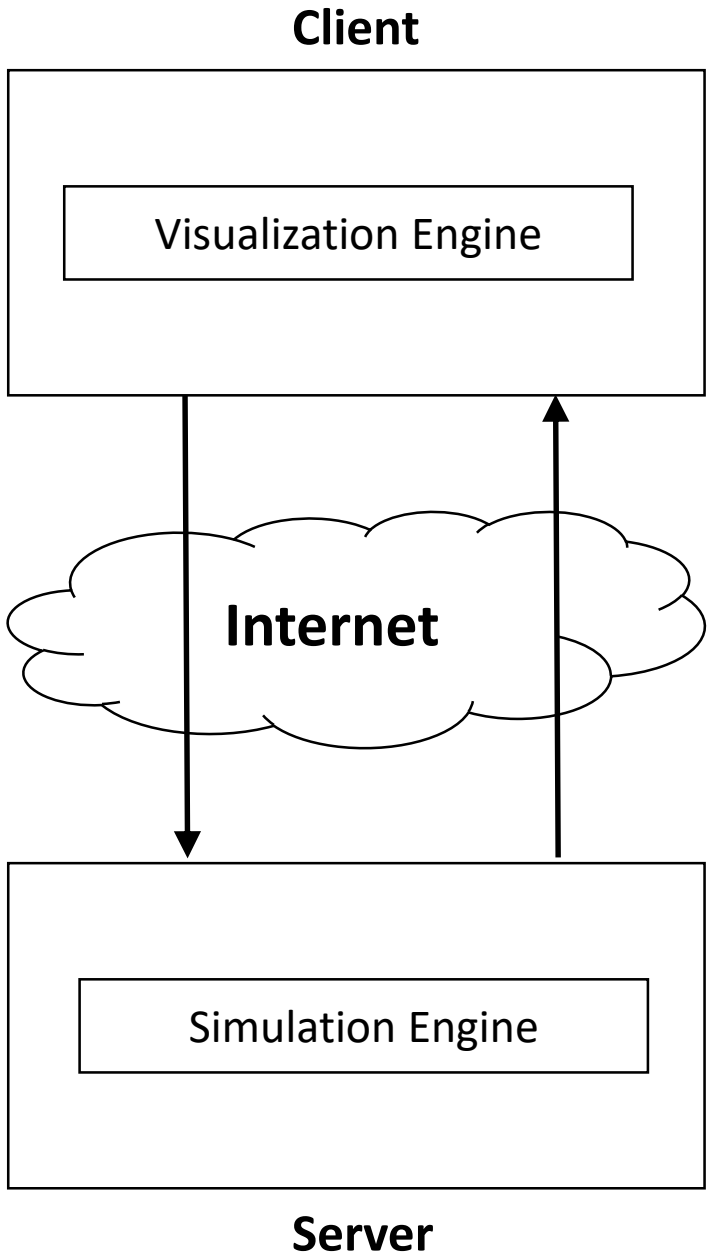
- Require **no communication** between the **client** and **server** after the initial simulation request phase
- The **simulation execution relies entirely on the server** computer's processing power and there is **no dependency on the communication infrastructure**
- Client is used to access the visual interface with the server, but it does not perform any real simulation task
- **Advantage**: Support to run larger simulations and easier maintenance
- **Disadvantage**: Users cannot observe dynamic processes at work or interrupt a running simulation in real-time



(Byrne et al. 2010)

Hybrid Architecture

- Combination of Local and Remote architectures best features
- The **server executes the simulation**, while the **client receives** and **displays** the simulation results
- **Client** is able to **observe** and **interact** with the simulation in **real-time**
- **Advantage**: Support to run larger simulations and easier maintenance
- **Disadvantage**: High volume of data exchange between the client and server during the simulation run



(Byrne et al. 2010)

Enabling Technologies

- **Adobe Flash** (*deprecated*) is a multimedia software platform used for production of rich Internet applications and embedded web browser video players
- **Java Applet** (*deprecated*) is a small application compiled to Java bytecode that is executed within a Java Virtual Machine in a process separate from the web browser
- **Web Services** is a web technology used for machine-to-machine communication, more specifically for transferring machine-readable file formats such as XML and JSON
- **JavaScript** is an interpreted programming language, one of the core technologies of the World Wide Web, that is executed by a dedicated JavaScript engine in the web browser
- **WebAssembly** is an open standard that defines a portable binary code format for executable programs and interfaces to facilitating interactions between such programs and their host environment

General-Purpose Platforms

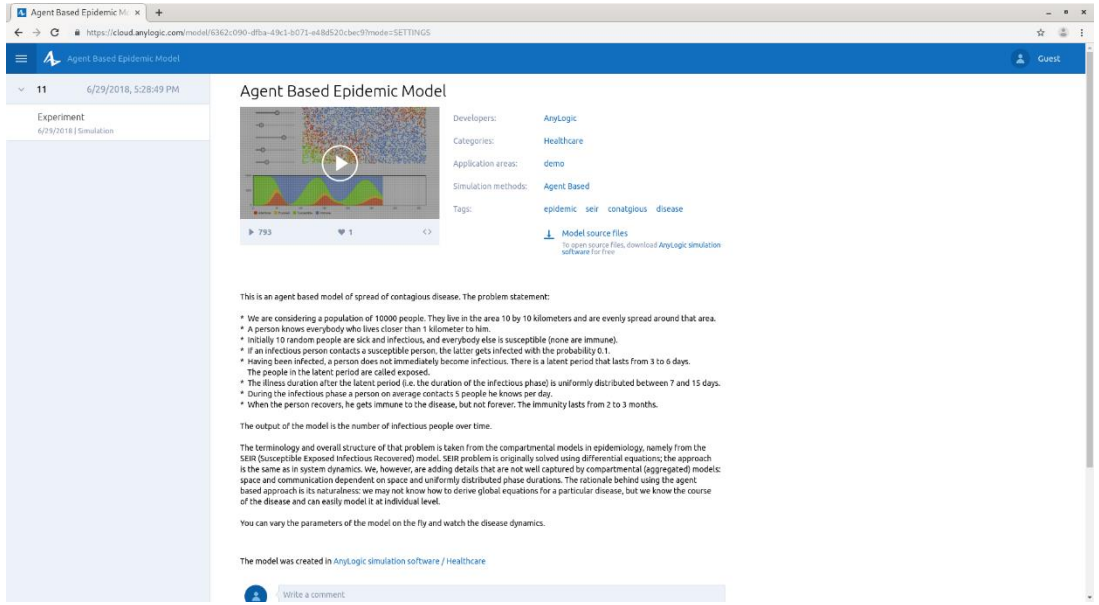
Despite the promising of the web-based simulation, there are few active general-purpose platforms available

- Anylogic Cloud
- NetLogo Web
- Insight Maker
- OESjs

AnyLogic Cloud

http://cloud.anylogic.com

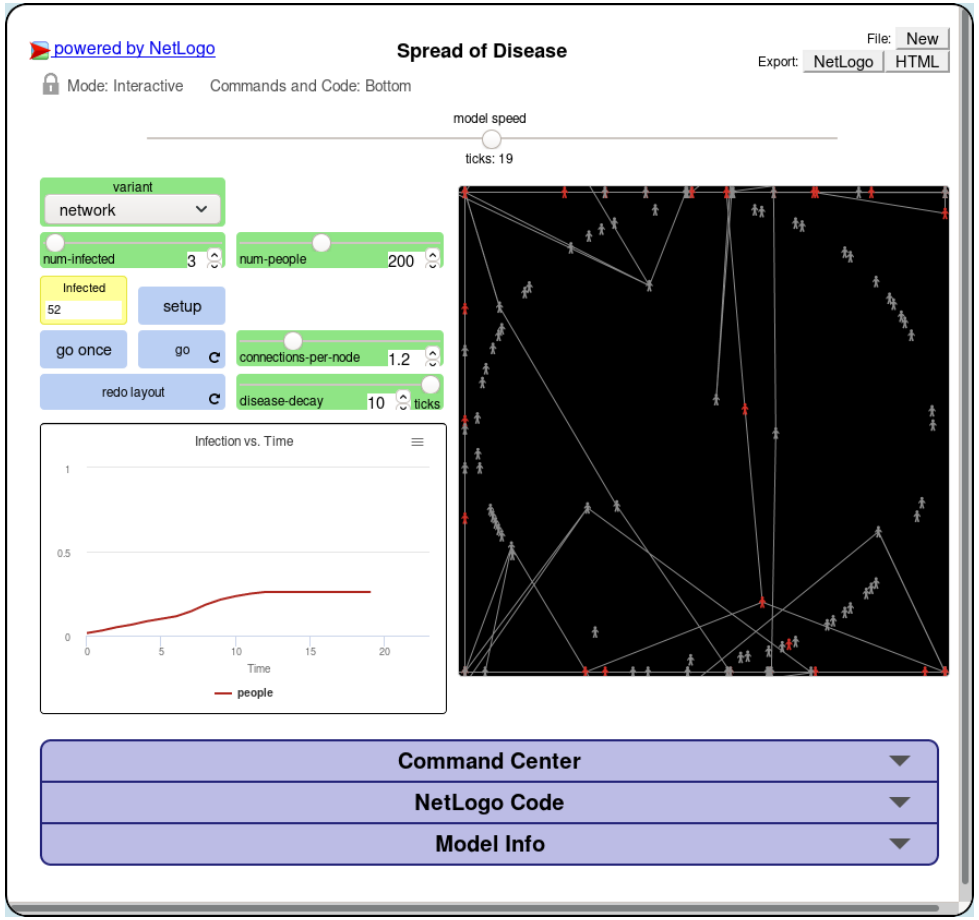
- AnyLogic Cloud is a web service for simulation analytics
- Allow users to store, access, run, and share simulation models online as well as analyze experiment results
- Enable to embed 2D and 3D AnyLogic model animation in HTML5 web pages
- The web interface retains allow users to change the speed of the model, watch 2D animation and navigate through 3D scenes
- Support System Dynamics, Agent Based and Discrete Event simulation and a combination of them



NetLogo Web

https://netlogoweb.org

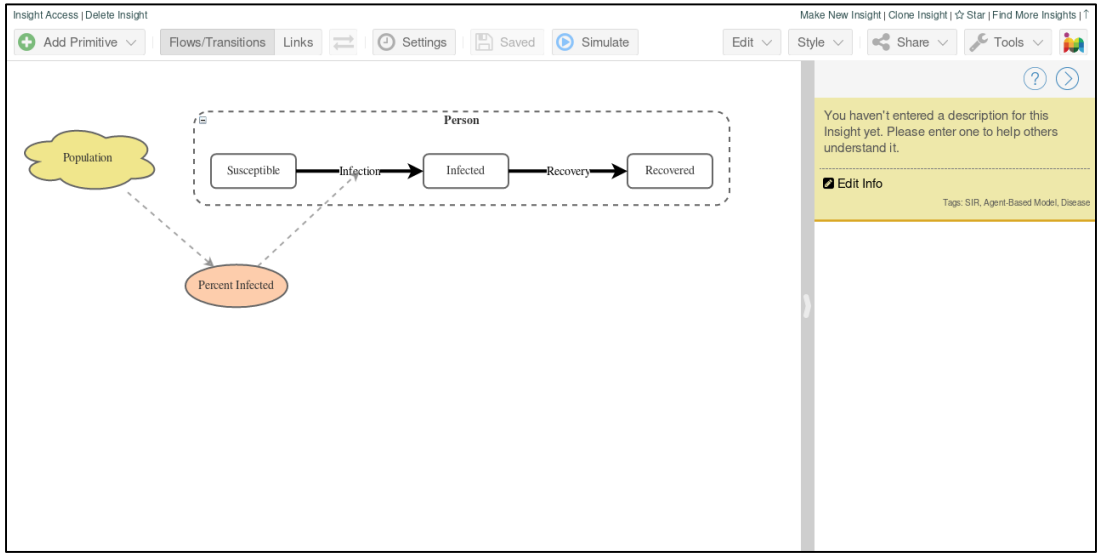
- A version of the NetLogo that runs entirely in the web browser
- NetLogo is a open source, free multi-agent programmable modeling environment fully programmable using a Logo dialect
- NetLogo Web does not support all features of the NetLogo yet
- Large library of built-in functions
- Allow to import and use models developed in NetLogo
- Support Agent Based simulation



Insight Maker

https://insightmaker.com

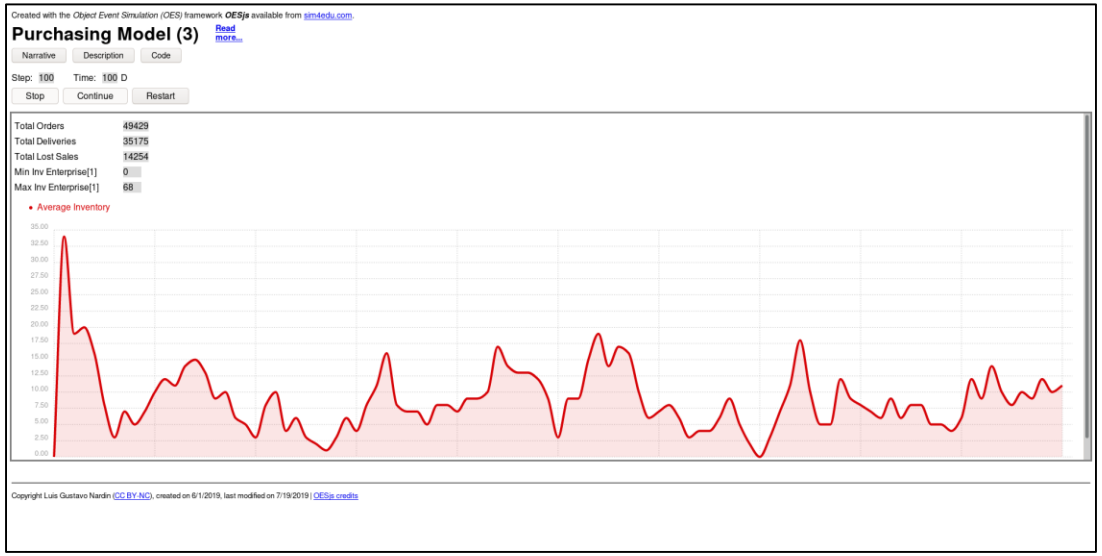
- Powerful open source, free simulation tool that runs in the web browser
- Provide a set of features for building, running and sharing simulation models
- Support extensive diagramming and modeling features that make easy the representation of a system
- Large library of built-in functions
- Support System Dynamics and Agent Based simulation



Object-Event Simulator JavaScript (OESjs)

https://sim4edu.com

- Open source and free **web-based simulator** developed using **open source technologies** (HTML, CSS, and JavaScript)
- **JavaScript-based simulation framework** implements the *Object-Event Simulation* paradigm, representing a general *Discrete Event Simulation* approach based on **object-oriented modeling** and **event scheduling**
- Supports two forms of simulations
  - **Standalone** simulation scenario
  - **Experiments**
- Support Discrete Event simulation



SIM4EDU
SIMULATION FOR EDUCATION

- The **Simulation for Education** (Sim4edu) project supports **web-based simulation** with **open source** technologies for **science and education**
- **Purposes**
  - provide technologies, such as **simulation libraries**, **frameworks**, and **simulators**, as well as a collection of simulation model examples
  - **facilitate building state-of-the-art** user interfaces without requiring simulation developers to learn all the recent web technologies involved (e.g., HTML5, CSS3, SVG and WebGL)

http://www.sim4edu.com

Performance

Tool	Licensing	Modeling	Architecture	Download (MB)	Memory (MB)	Execution (MB)
AnyLogic Cloud	Commercial	Offline	Hybrid	0.68	2.80	22.00+
Insight Maker	Free	Online	Local	1.50	22.50	~0.00
NetLogo Web	Free	Online, Offline	Local	1.20	34.10	~0.00
OESjs	Free	Offline	Local	0.16	2.90	~0.00

Results generated using the same underlying epidemiology SIR (Susceptible-Infected-Recovered) model

- AnyLogic Cloud – Agent Based Epidemic Model
  https://cloud.anylogic.com/model/6362c090-dfba-49c1-b071-e48d520cbec9?mode=SETTINGS
- Inishight Maker – SIR Model
  https://insightmaker.com/insight/156381/SIR-Model
- NetLogo Web – Spread of Disease
  https://netlogoweb.org/launch#https://netlogoweb.org/assets/modelslib/IABM%20Textbook/chapter%206/Spread%20of%20Disease.nlogo
- OESjs – Susceptible-Infectious-Recovered (SIR) Model
  https://sim4edu.com/sims/25/index.html

Advantages

- Easy of use
- Collaboration
- License and deployment model
- Model reuse
- Cross-platform capability
- Controlled access
- Wide availability
- Versioning, customization and maintenance
- Integration and interoperability

(Byrne et al. 2010)

Disadvantages

- Loss of speed
- Graphical user interface limitations
- Security vulnerability
- Web-based simulation application stability
- Licensing restriction
- Difficulty in simplifying simulation

(Byrne et al. 2010)

JavaScript-Based Simulation with OESjs

Introduction to JavaScript

- JavaScript was developed in May 1995 by *Brendan Eich*
- JavaScript is a **scripting language**
  - Lightweight programming language
  - Programming code is embedded in HTML
  - Can be executed by all modern web browsers
- JavaScript is **weakly typed** and **dynamic**
- JavaScript is **object-oriented**, but in a *different* way than *classical OO languages*, e.g.,
  - objects can be created directly without instantiating any class
  - properties can be added to an object or class definition at run-time

Object-Event Simulator JavaScript (OESjs)

- **JavaScript-based simulation framework** that implements the *Object-Event Simulation* (OES) paradigm
- The models is composed of two basic types of entities
  - Object Types
  - Event Types
- OESjs supports two forms of simulations
  - **Standalone** simulation scenario
  - Simulation **experiments**, which define a set of simulation scenarios by defining value sets for certain model variables, such that an experiment run consists of a set of scenario runs

Purchasing Model Description

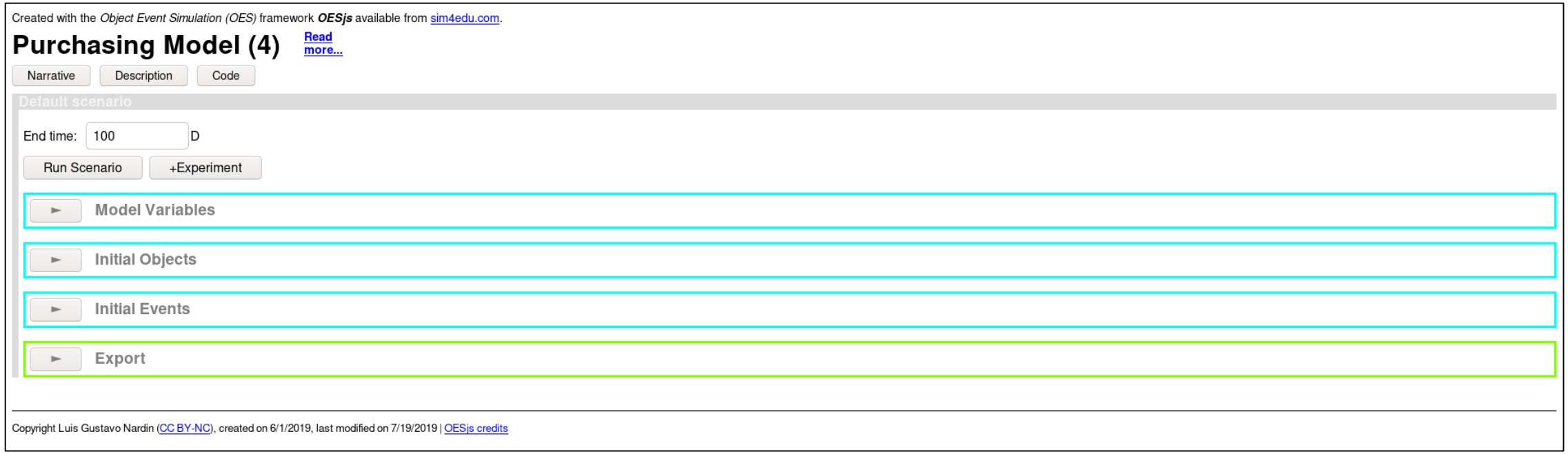
- The model represents a simple purchase order transaction of a single type of item between Enterprises and Consumers
- On each day, Enterprises produce a quantity of a single type of item and update the unit item price
- On each day, Consumers decide whether to order new items. If so, they decide which Enterprise to purchase from the Enterprise with the lowest item price in its list of preferred Enterprises and the quantity of items to purchase
- If the ordered quantity of items is in stock, the Enterprise delivers the items to the Consumer
- Otherwise, the Enterprise delivers the quantity of products it has in stock and registers the remaining quantity as lost sales

Purchasing Model

- Available at https://github.com/gnardin/PurchasingModel/
- Download
  - Release https://github.com/gnardin/PurchasingModel/releases/tag/SSC19

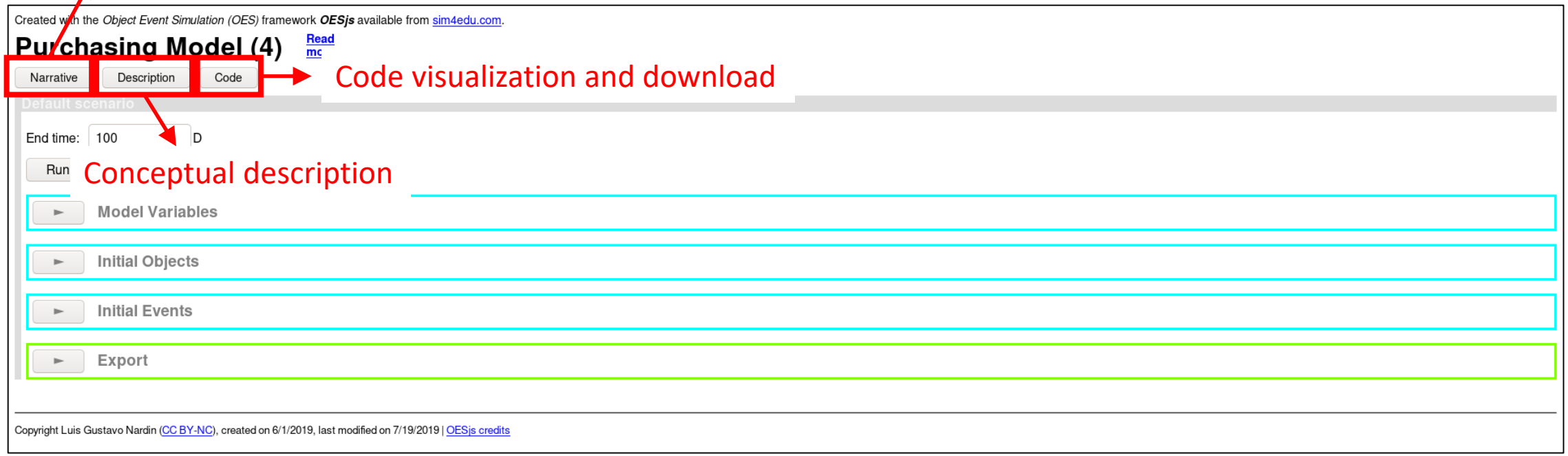
    or
  - Clone `git clone https://github.com/gnardin/PurchasingModel.git`
- Navigate to the `PurchasingModel/OESjs/01` folder
- Open the `simulation.html` using Firefox
- Click on `Run Scenario` to execute the simulation

Web Interface

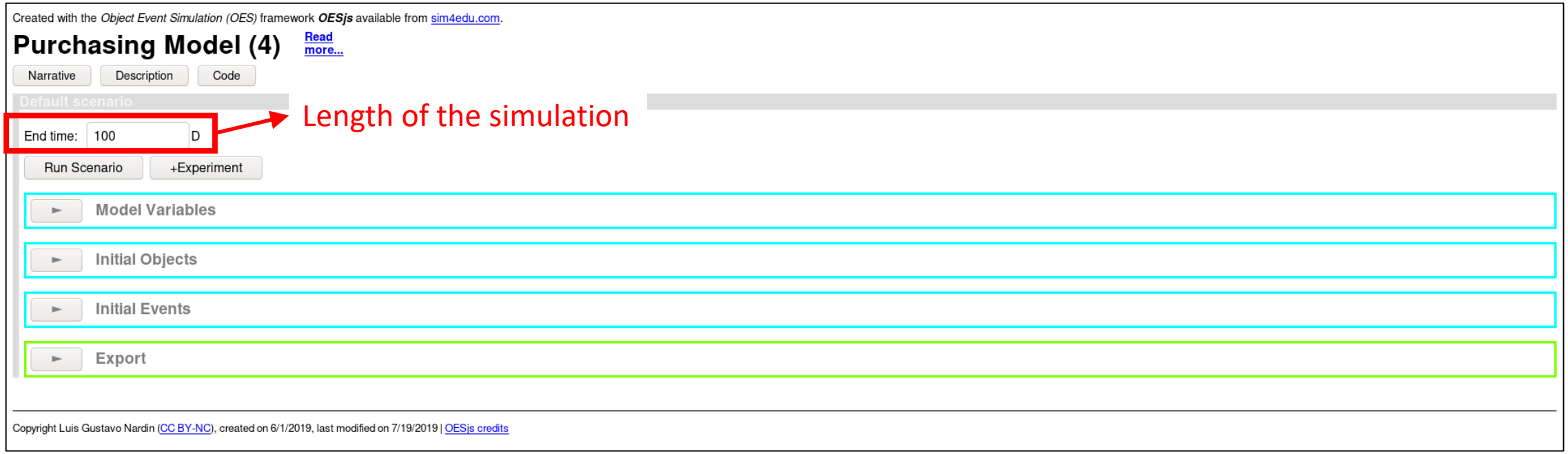


Web Interface

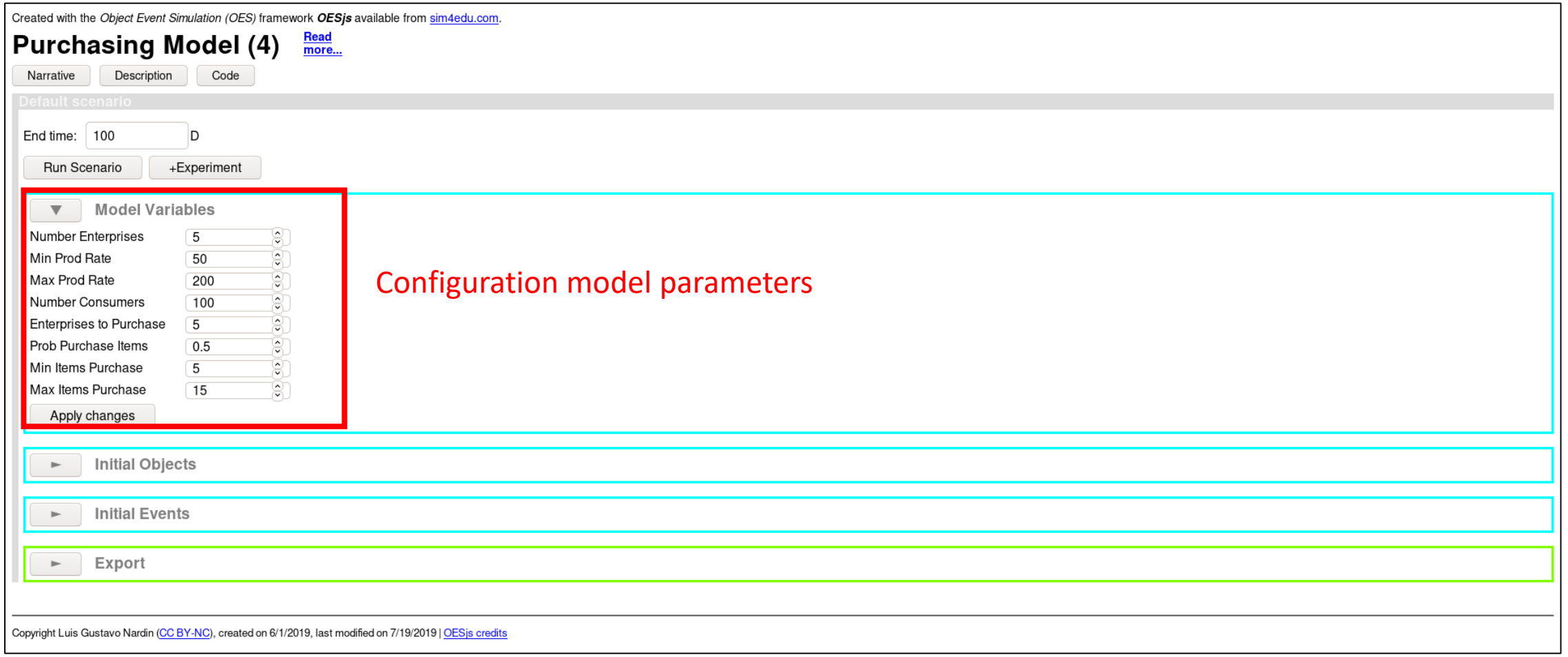
Narrative description



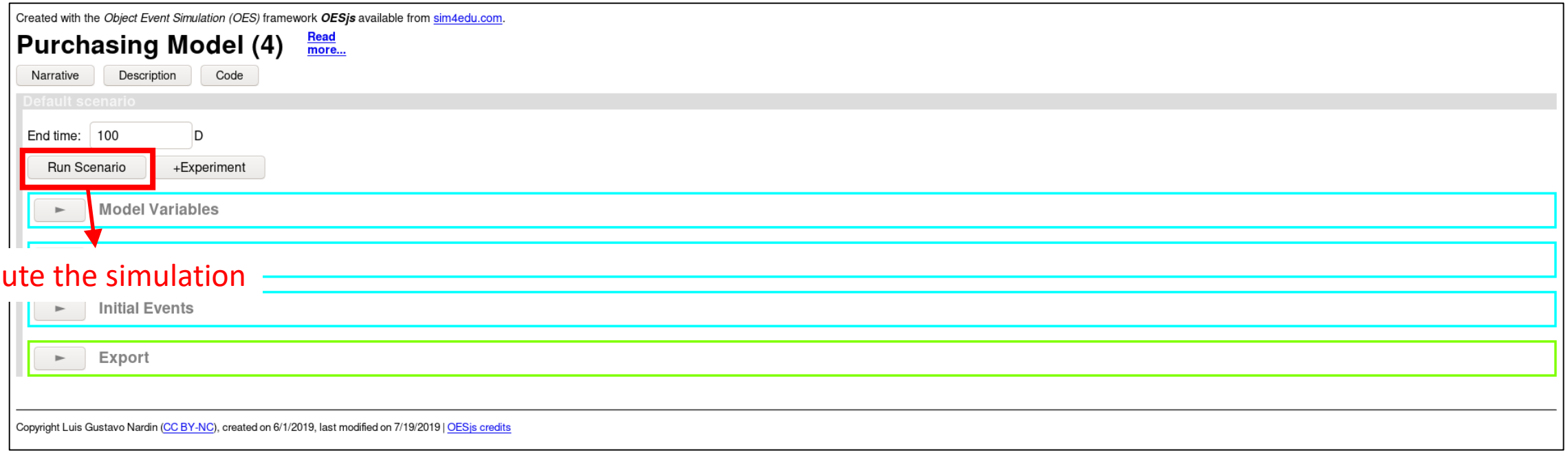
Web Interface



Web interface



Web Interface



Execute the simulation

Web Interface

Created with the *Object Event Simulation (OES)* framework ***OESjs*** available from sim4edu.com.

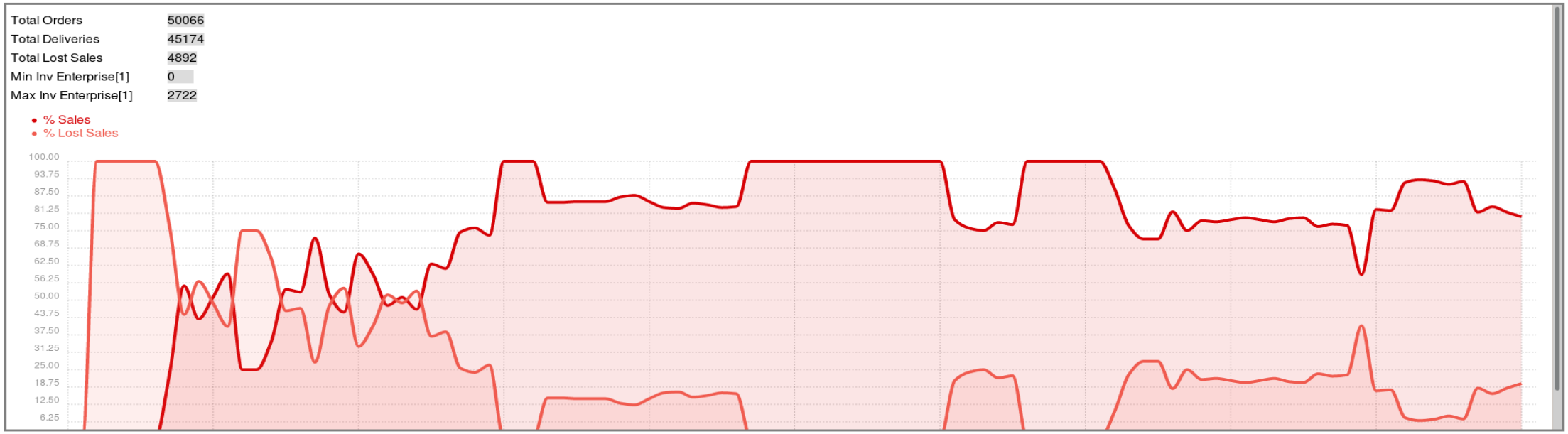
Purchasing Model (4) Read more...

Narrative | Description | Code

Step: 100 Time: 100 D

Stop | Continue | Restart

Total Orders	50066
Total Deliveries	45174
Total Lost Sales	4892
Min Inv Enterprise[1]	0
Max Inv Enterprise[1]	2722



Files and Folder Structure

Framework Folder/File	Description
loadManager.js	Load the simulator and the necessary simulation model files
loadManagerConfig.js	Configuration file for the Load manager
framework/	Contain the OESjs framework source-code

Simulation Model Folders	Description
01	Basic purchasing simulation model containing only object types
02	Purchasing simulation model containing object and event types
03	Purchasing simulation model composed of multiple instances of Enterprise and Consumer
04	Purchasing simulation model in which the Consumer makes a more elaborate decision from whom to purchase products

Simulation Model Files

File	Description
simulation.html	Launch the simulation model
description.html	Contain the description of the simulation model
metadata.js	Contain the meta-data information about the simulation model
simulation-worker.js	Responsible for loading the simulation model in the Web worker
simulation.js	Define the simulation model
…	Dependent on the model implemented, usually they are Object and Event Type class definitions

Meta-data Information (metadata.js)

- Contains the meta-data information about the Simulation model

Property	Description
`sim.model.name`	Name of the simulation model
`sim.model.title`	Title of the simulation model
`sim.model.systemNarrative`	Description of the system modeled
`sim.model.shortDescription`	Short description of the simulation model
`sim.model.source`	Citation of the document where the model is officially defined
`sim.model.license`	License associated to the simulation model
`sim.model.creator`	Name of the creator(s) of the simulation model
`sim.model.created`	Date of creation
`sim.model.modified`	Date of last modification

Simulation Time Progression (simulation.js)

- Time progression supported
  - Discrete
  - Continuous
- **Discrete** time progression
  - **fixed-increment** time progression
  - **next-event** time progression
- A model with pure fixed-increment time progression defines an `OnEachTimeStep` procedure and a `timeIncrement` parameter

Property	Description
`sim.model.time`	Define the time model (i.e., "`discrete`" or "`continuous`")
`sim.model.timeUnit`	Define the model's time unit (i.e., "`ms`", "`s`", "`m`", "`h`", "`D`", "`W`", "`M`" or "`Y`")
`sim.model.timeIncrement`	Support the fixed-time increment time progression used to trigger the `onEventTimeStep` function

Scenario Parameters (simulation.js)

- Define the default simulation parameters

Property	Description
`sim.scenario.simulationEndTime`	Length of the simulation
`sim.scenario.randomSeed`	Seed of the random generator

Purchasing Model – Version 1

https://gnardin.github.io/PurchasingModel/01.html

- Single Enterprise produces and sells one single type of item
- Single Consumer purchases items from the Enterprise
- On each day
  - the Enterprise produces a finite quantity of items
  - the Consumer orders a specific quantity of items from the Enterprise
    - If the ordered quantity is in stock, the Enterprise delivers the quantity items to the Consumer and the order is fulfilled
    - Otherwise, the Enterprise delivers the quantity of items in stock to the Consumer (i.e., order is partially fulfilled) and registers the non-fulfilled quantity as lost sales

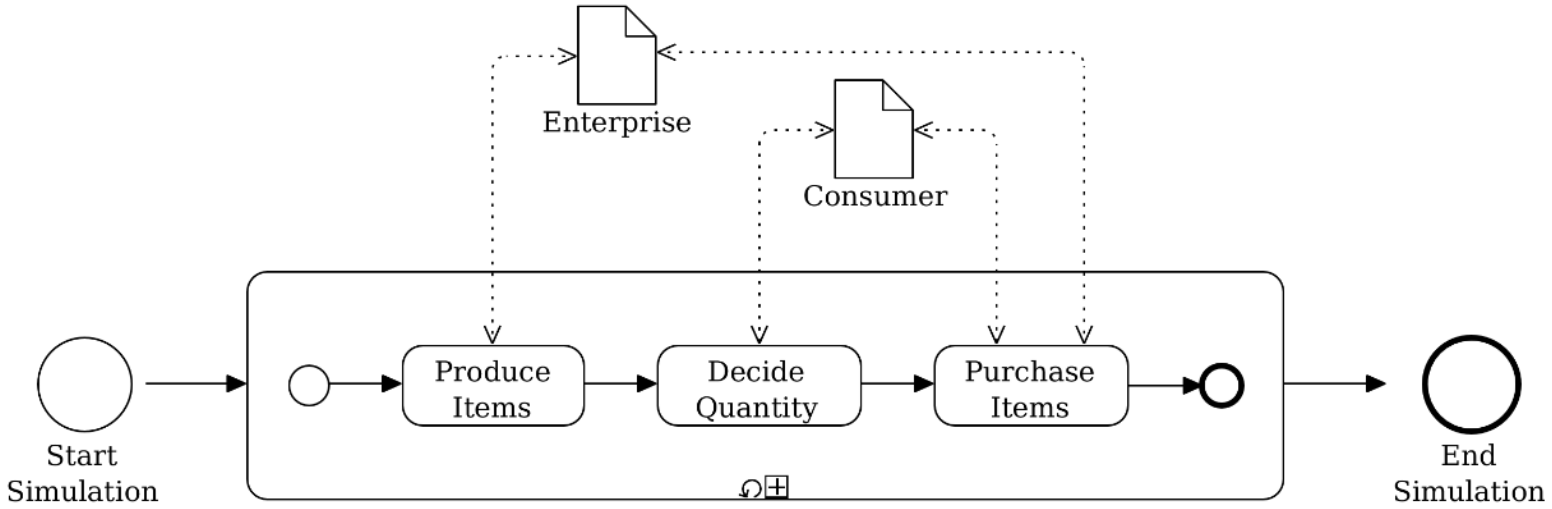
Purchasing Model – Version 1

https://gnardin.github.io/PurchasingModel/01.html

<<object type>> **Consumer**
purchaseMin : NonNegativeInteger purchaseMax : NonNegativeInteger
decideOrder() : NonNegativeInteger

<<object type>> **Enterprise**
inventoryLevel : NonNegativeInteger productionRateMin : NonNegativeInteger productionRateMax : NonNegativeInteger
produceItems()

Class diagram



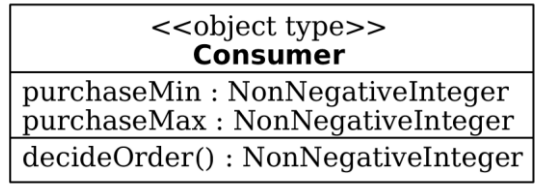
BPMN diagram

Object Types

- Object types are defined in the form of classes
- Each object type should reside in a file with the same name

Consumer Object Type class

```
var Consumer = new cLASS( {
  Name: "Consumer",
  shortLabel: "Consumer",
  supertypeName: "oBJECT",

  properties: {
    "purchaseMin": { range: "NonNegativeInteger", label: "Min Items Purchase" },
    "purchaseMax": { range: "NonNegativeInteger", label: "Max Items Purchase" }
  },

  methods: {
    "decideOrder": function () {
      return rand.uniformInt( this.purchaseMin, this.purchaseMax );
    }
  }
} );
```



Data Types

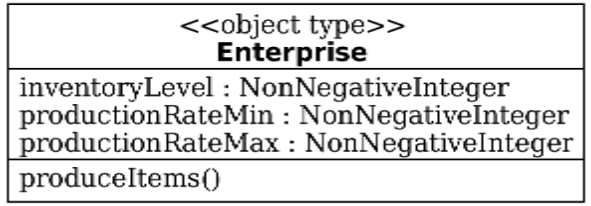
Data Type	Description
String	String
NonEmptyString	Non-empty string
Integer	Integer
NonNegativeInteger	Positive Integer including 0
PositiveInteger	Positive Integer excluding 0
Decimal	Decimal
Boolean	Boolean
Object Type class	Any object type class defined in the simulation model

Object Types

- Object types are defined in the form of classes
- Each object type should reside in a file with the same name

Enterprise Object Type class

```
var Enterprise = new cLASS( {
  Name: "Enterprise",
  shortLabel: "Enterprise",
  supertypeName: "oBJECT",

  properties: {
    "inventoryLevel": { range: "NonNegativeInteger", label: "Inventory Level" },
    "productionRateMin": { range: "NonNegativeInteger", label: "Min Prod Rate" }
    "productionRateMax": { range: "NonNegativeInteger", label: "Max Prod Rate" }
  },

  methods: {
    "produceItems": function () {
      this.inventoryLevel += rand.uniformInt( this.productionRateMin,
        this.productionRateMax );
    }
  }
} );
```



Probability Distribution Functions

Probability Distribution Function	OESjs Library Method	Example
Uniform	`uniform(lower, upper)` `uniform()`	`rand.uniform( 0.5, 1.5 )` `rand.uniform()`
Discrete Uniform	`uniformInt(lower, upper)`	`rand.uniformInt( 1, 6 )`
Triangular	`triangular(lower, upper, mode)`	`rand.triangular( 0.5, 1.5, 1.0 )`
Exponential	`exponential(rate)`	`rand.exponential( 0.5 )`
Gamma	`gamma(shape, scale)`	`rand.gamma( 1.0, 2.0 )`
Normal	`normal(mean, stdDev)`	`rand.normal( 1.5, 0.5 )`
Paretto	`paretto(shape)`	`rand.paretto( 2.0 )`
Weibull	`weibull(scale, shape)`	`rand.weibull( 1, 0.5 )`

Simulation Model (simulation.js)

Property	Description
`sim.model.objectTypes`	Define the list of object types
`sim.model.eventTypes`	Define the list of event types

```
/* Object and Event types */
sim.model.objectTypes = [ "Enterprise", "Consumer" ];
sim.model.eventTypes = [];
```

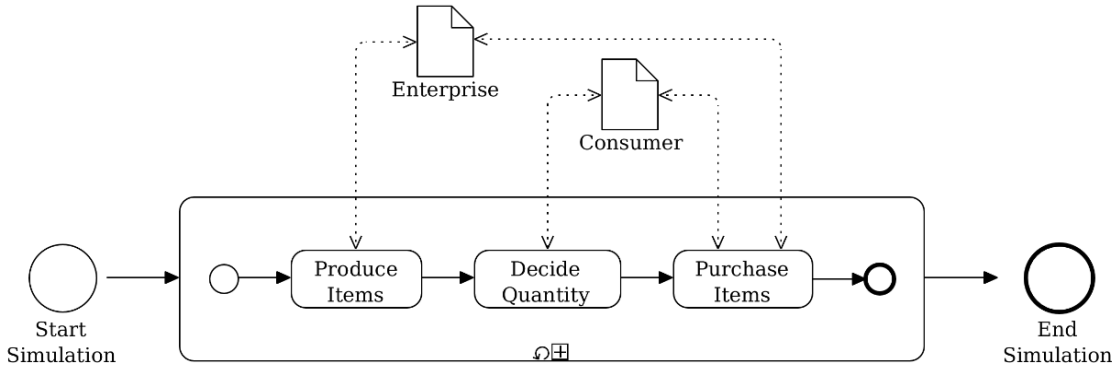
Simulation Initial State (simulation.js)

- Define the objects created
- Define the events scheduled

```
// Initial Objects
sim.scenario.initialState.objects = {
  "1": { typeName: "Enterprise", name: "enterprise", inventoryLevel: 0, productionRateMin: 5,
         productionRateMax: 15 },
  "2": { typeName: "Consumer",  name: "consumer", purchaseMin: 5, purchaseMax: 15 }
};

//Initial Events
sim.scenario.initialState.events = [];

//Initial Functions
sim.scenario.setupInitialState = function () {}
```

Simulation Process (simulation.js)

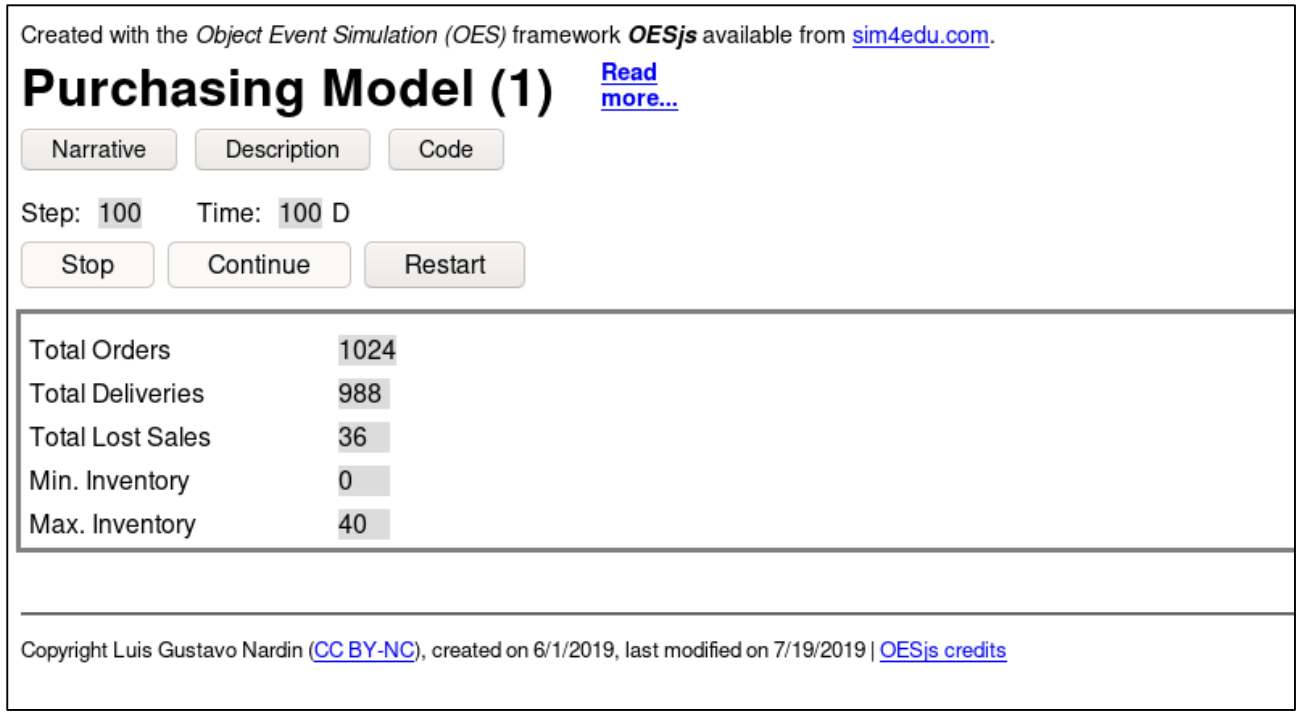
```
sim.model.OnEachTimeStep = function () {
  var quantity;
  var enterprise = cLASS[ "Enterprise" ].instances[ "1" ];
  var consumer = cLASS[ "Consumer" ].instances[ "2" ];

  // Enterprise produces items
  enterprise.produceItems();

  // Consumer defines the quantity of items to order
  quantity = consumer.decideOrder();

  sim.stat.itemsOrdered += quantity;

  if ( enterprise.inventoryLevel >= quantity ) {
    enterprise.inventoryLevel -= quantity;

    sim.stat.itemsDelivered += quantity;
  } else {
    sim.stat.lostSales += quantity - enterprise.inventoryLevel;
    sim.stat.itemsDelivered += enterprise.inventoryLevel;

    enterprise.inventoryLevel = 0;
  }
}
```



BPMN diagram

Output Statistics (simulation.js)

```
sim.model.statistics = {
  ...
  "lostSales": {
    range: "NonNegativeInteger",
    label: "Total Lost Sales",
    initialValue: 0
  },
  "minInventory": {
    objectType: "Enterprise",
    objectIdRef: 1,
    property: "inventoryLevel",
    aggregationFunction: "min",
    label: "Min. Inventory"
  },
  "maxInventory": {
    objectType: "Enterprise",
    objectIdRef: 1,
    property: "inventoryLevel",
    aggregationFunction: "max",
    label: "Max. Inventory"
  }
};
```



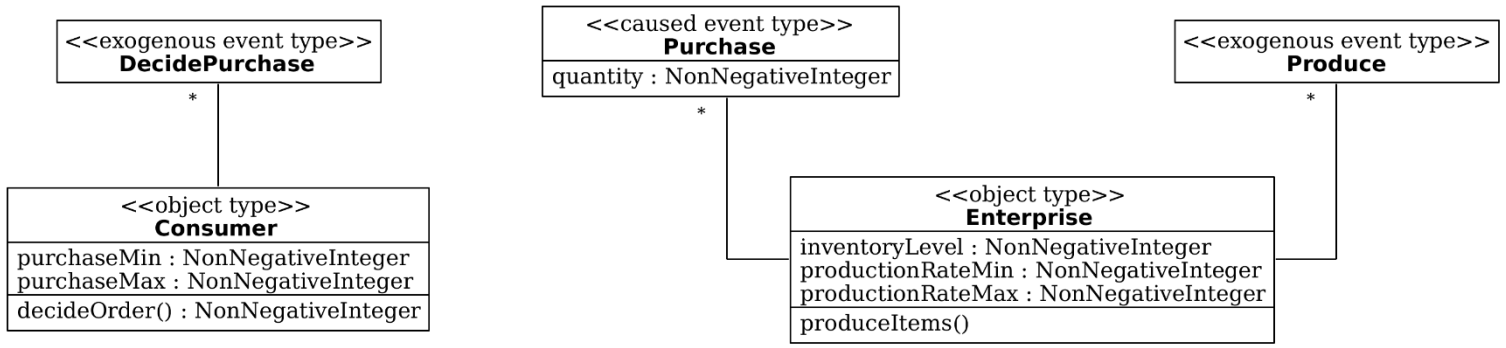
Purchasing Model – Version 2

https://gnardin.github.io/PurchasingModel/02.html

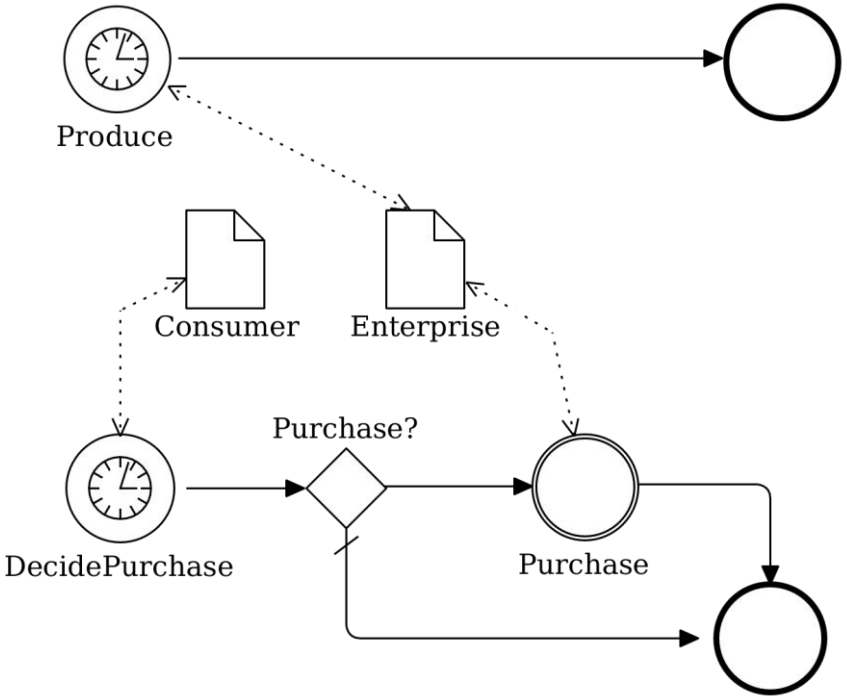
- Single Enterprise produces and sells one single type of item
- Single Consumer purchases items from the Enterprise
- On each day
  - the Enterprise **generates a Produce event** to produce a finite quantity of items
  - the Consumer **generates a DecidePurchase event** that generate a **Purchase event** with **50% probability** to order a specific quantity of items from the Enterprise
    - If the ordered quantity is in stock, the Enterprise delivers the quantity items to the Consumer and the order is fulfilled
    - Otherwise, the Enterprise delivers the quantity of items in stock to the Consumer (i.e., order is partially fulfilled) and registers the non-fulfilled quantity as lost sales

Purchasing Model – Version 2

https://gnardin.github.io/PurchasingModel/02.html



Class diagram



BPMN diagram

Event Types

- Event types are defined in the form of classes
- We distinguish between two kinds of events
  - **caused events** are caused by other events occurring during a simulation run
  - **exogenous events** are caused by factors external to the simulation model

Exogenous Event Type

```
var DecidePurchase = new cLASS({
  Name: "DecidePurchase",
  supertypeName: "eVENT",
  properties: {
    "consumer": { range: "Consumer" }
  },
  methods: {
    "onEvent": function () {
      ...
    }
  }
});

DecidePurchase.priority = 0;

DecidePurchase.recurrence = function () {
  return 1;
};
```

```
"onEvent": function () {
    var followupEvents = [];

    // Quantity to purchase
    var quantity = this.consumer.decideOrder();

    // Whom to purchase from
    var enterprise = cLASS[ "Enterprise" ].instances[ "1" ];

    followupEvents.push( new Purchase( {
      occTime: this.occTime + 1,
      enterprise: enterprise,
      quantity: quantity
    } ) );

    return followupEvents;
  }
}
```

Event Rule

Caused Event Type

```
var Purchase = new cLASS({
  Name: "Purchase",
  supertypeName: "eVENT",
  properties: {
    "enterprise": { range: "Enterprise" }
  },
  methods: {
    "onEvent": function () {
      ...
    }
  }
});

Purchase.priority = 2;
```

```
"onEvent": function () {
    var followupEvents = [];

    sim.stat.itemsOrdered += this.quantity;

    if ( this.enterprise.inventoryLevel >= this.quantity ) {
      this.enterprise.inventoryLevel -= this.quantity;

      sim.stat.itemsDelivered += this.quantity;
    } else {
      sim.stat.itemsDelivered += this.enterprise.inventoryLevel;
      sim.stat.lostSales += this.quantity -
                            this.enterprise.inventoryLevel;

      this.enterprise.inventoryLevel = 0;
    }

    return followupEvents;
  }
}
```

Event Rule

Simulation Model (simulation.js)

Property	Description
`sim.model.objectTypes`	Define the list of object types
`sim.model.eventTypes`	Define the list of event types

```
/* Object and Event types */
sim.model.objectTypes = [ “Enterprise", “Consumer" ];
sim.model.eventTypes = [ “DecidePurchase”, “Purchase”, “Produce” ];
```

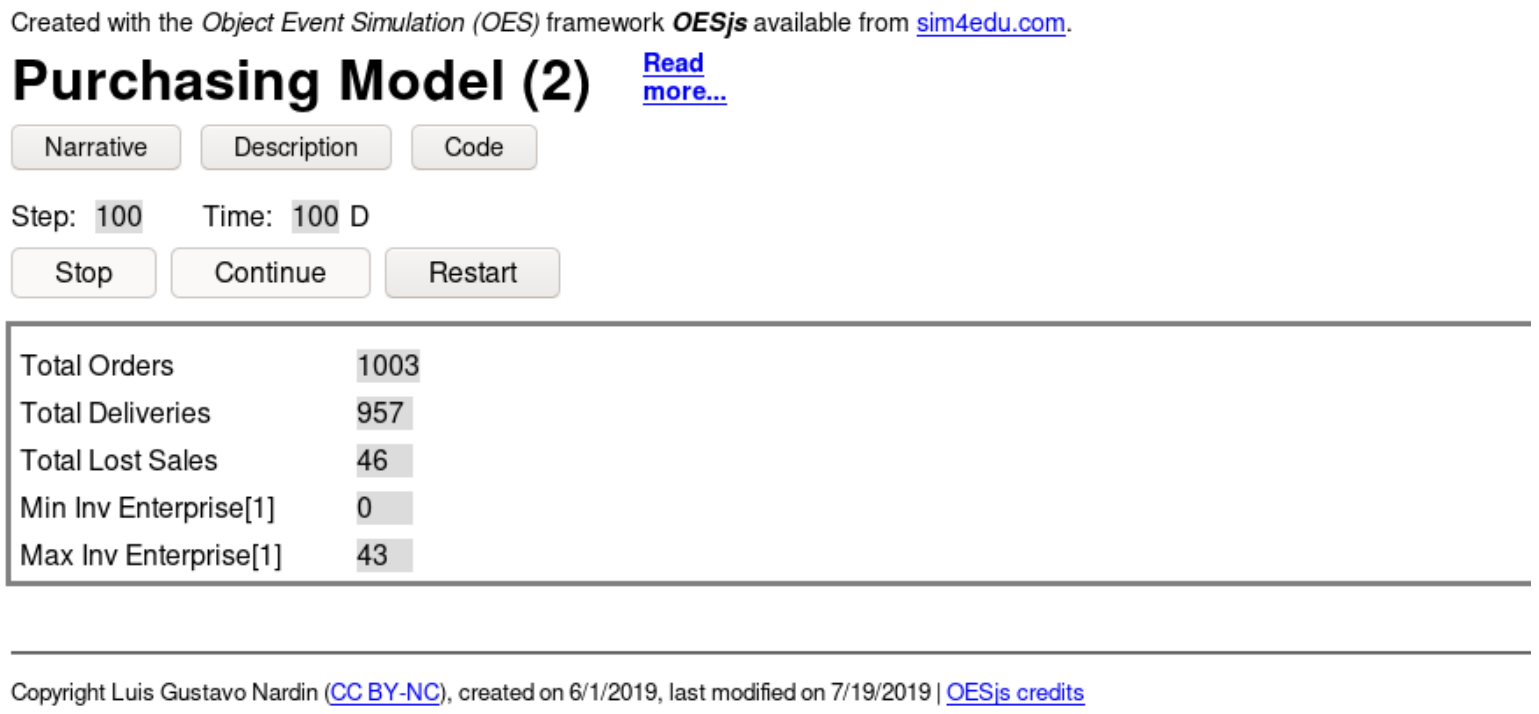
Simulation Initial State (simulation.js)

- Define the objects created
- Define the events scheduled

```
// Initial Objects
sim.scenario.initialState.objects = {
  "1": { typeName: "Enterprise", name: "enterprise", inventoryLevel: 0, productionRateMin: 5,
         productionRateMax: 15 },
  "2": { typeName: "Consumer",  name: "consumer", purchaseMin: 5, purchaseMax: 15 }
};

//Initial Events
sim.scenario.initialState.events = [
  { typeName: "Produce", occTime: 1, enterprise: 1 },
  { typeName: "DecidePurchase",  occTime: 1, consumer: 2 }
];

//Initial Functions
sim.scenario.setupInitialState = function () {}
```

Purchasing Model – Version 2

https://gnardin.github.io/PurchasingModel/02.html



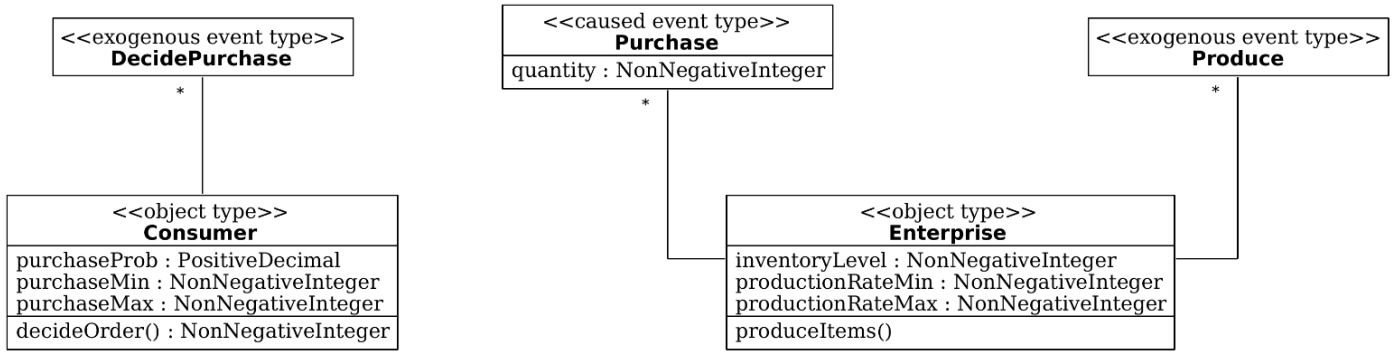
Purchasing Model – Version 3

https://gnardin.github.io/PurchasingModel/03.html

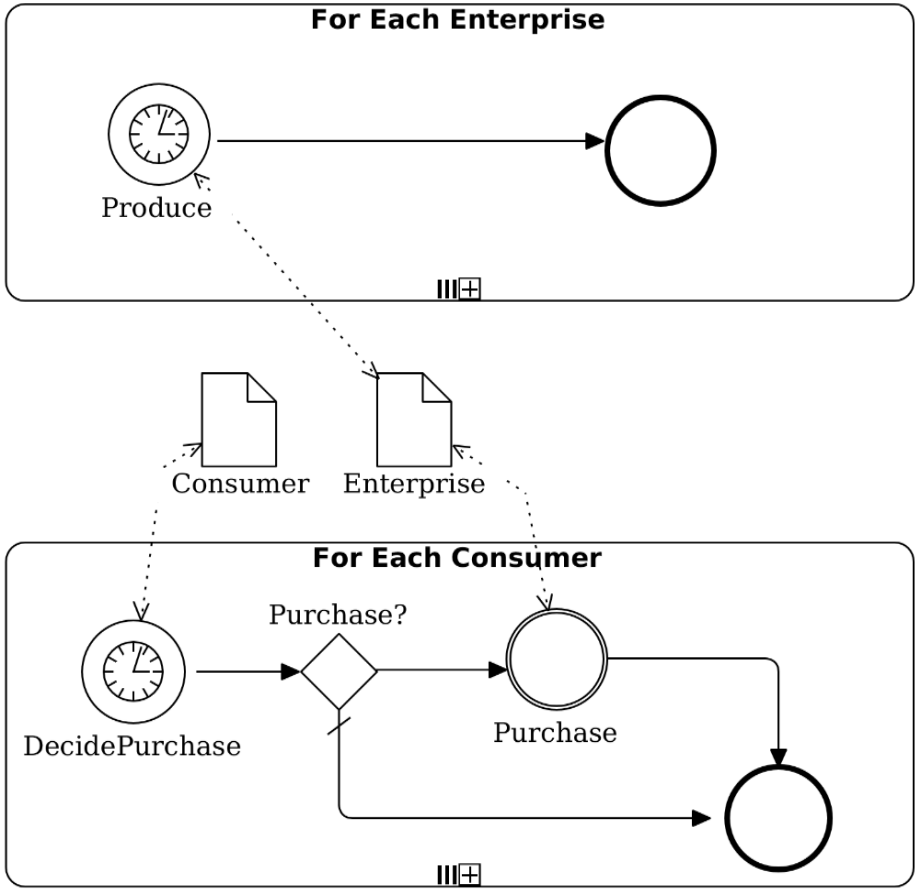
- **Multiple Enterprises** produce and sell one single type of item
- **Multiple Consumers** purchase items from Enterprises
- On each day
  - Enterprises **generate a Produce event** to produce a finite quantity of items
  - Consumers **generates a DecidePurchase event** that generates a **Purchase event** with a **probability** to order a specific quantity of items from one Enterprise
    - If the ordered quantity is in stock, the Enterprise delivers the quantity items to the Consumer and the order is fulfilled
    - Otherwise, the Enterprise delivers the quantity of items in stock to the Consumer (i.e., order is partially fulfilled) and registers the non-fulfilled quantity as lost sales

Purchasing Model – Version 2

https://gnardin.github.io/PurchasingModel/02.html



Class diagram



BPMN diagram

Global Variables (simulation.js)

```
/* Global Variables */
sim.model.v.nmrOfEnterprises = {
  range: "NonNegativeInteger",
  initialValue: 10,
  label: "Number Enterprises",
  hint: "The number of enterprises"
};
sim.model.v.purchaseProb = {
  range: "Decimal",
  initialValue: 0.5,
  decimalPlaces: 2,
  label: "Prob Purchase Items“,
  hint: "Probability of consumer purchase items"
};
```

- To refer to these variables in the simulation code use `sim.v.<name>` instead of `sim.model.v.<name>`

Simulation Initial State (simulation.js)

```
// Initial Objects
sim.scenario.initialState.objects = {};

//Initial Events
sim.scenario.initialState.events = [];

//Initial Functions
sim.scenario.setupInitialState = function () {
  var i, objId;

  for(i = 1; i <= sim.v.nmrOfEnterprises; i += 1) {
    objId = i;
    sim.addObject( new Enterprise( {
      id: objId,
      name: "enterprise" + i,
      inventoryLevel: 0,
      productionRateMin: sim.v.productionRateMin,
      productionRateMax: sim.v.productionRateMax
    } ) );

    sim.scheduleEvent( new Produce( {
      occTime: 1,
      enterprise: objId
    } ) );
  }

  for (; i <= sim.v.nmrOfEnterprises+sim.v.nmrOfConsumers; i+=1) {
    objId = i;
    sim.addObject( new Consumer( {
      id: objId,
      name: "consumer" + ( i - sim.v.nrmOfEnterprises + 1),
      purchaseProb: sim.v.purchaseProb,
      itemPurchaseMin: sim.v.purchaseMin,
      itemPurchaseMax: sim.v.purchaseMax
    } ) );

    sim.scheduleEvent( new DecidePurchase( {
      occTime: 1,
      consumer: objId
    } ) );
  }
};
```

Output Statistics (simulation.js)

```
sim.model.statistics = {
  ...
  "avgInventory": {
    range: "PositiveInteger",
    label: "Average Inventory",
    initialValue: 0,
    showTimeSeries: true,
    computeOnlyAtEnd: false,
    expression: function () {
      var total = 0;
      var enterprises = cLASS[ "Enterprise" ].instances;
      Object.keys( enterprises ).forEach( function ( objId ) {
        total += enterprises[ objId ].inventoryLevel;
      } );

      return total / Object.keys( enterprises ).length;
    }
  }
};
```

Purchasing Model – Version 3

https://gnardin.github.io/PurchasingModel/03.html

Created with the *Object Event Simulation (OES)* framework ***OESjs*** available from sim4edu.com.

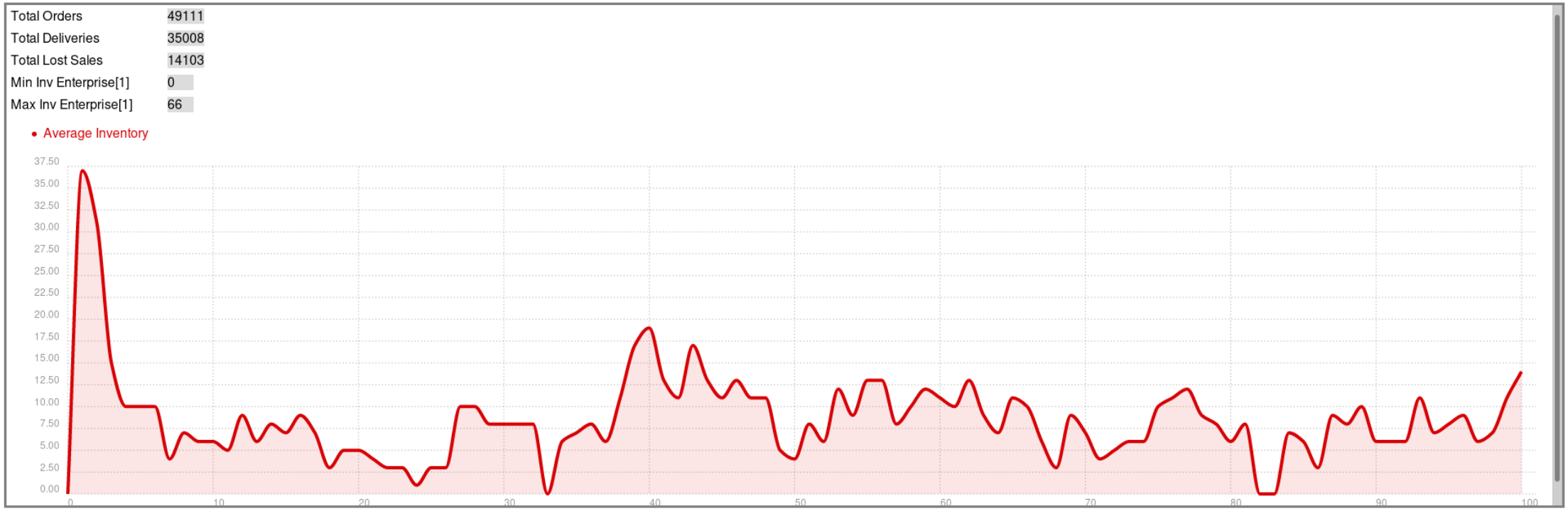
Purchasing Model (3) Read more...

Narrative | Description | Code

Step: 100 Time: 100 D

Stop | Continue | Restart

Total Orders	49111
Total Deliveries	35008
Total Lost Sales	14103
Min Inv Enterprise[1]	0
Max Inv Enterprise[1]	66



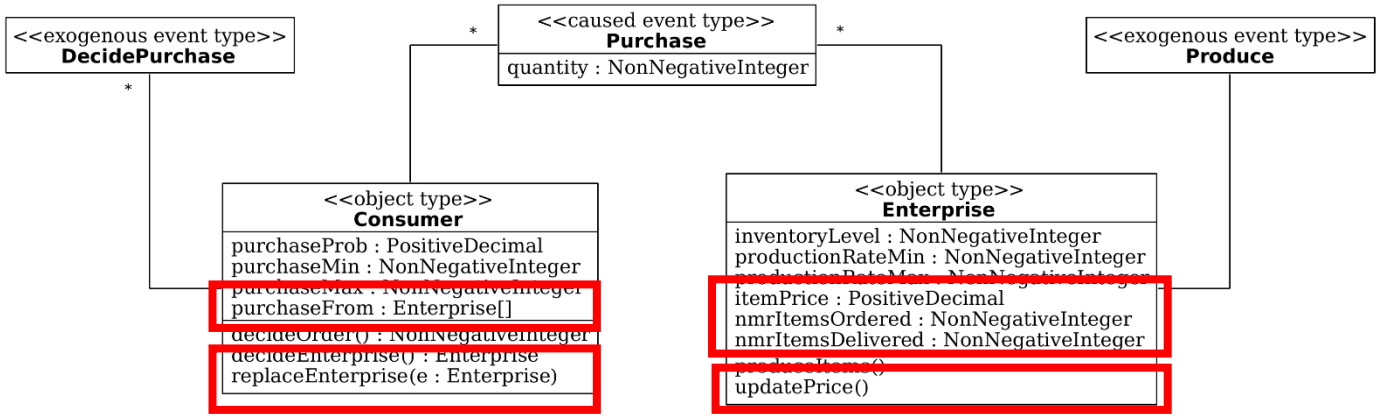
Purchasing Model – Version 4

https://gnardin.github.io/PurchasingModel/04.html

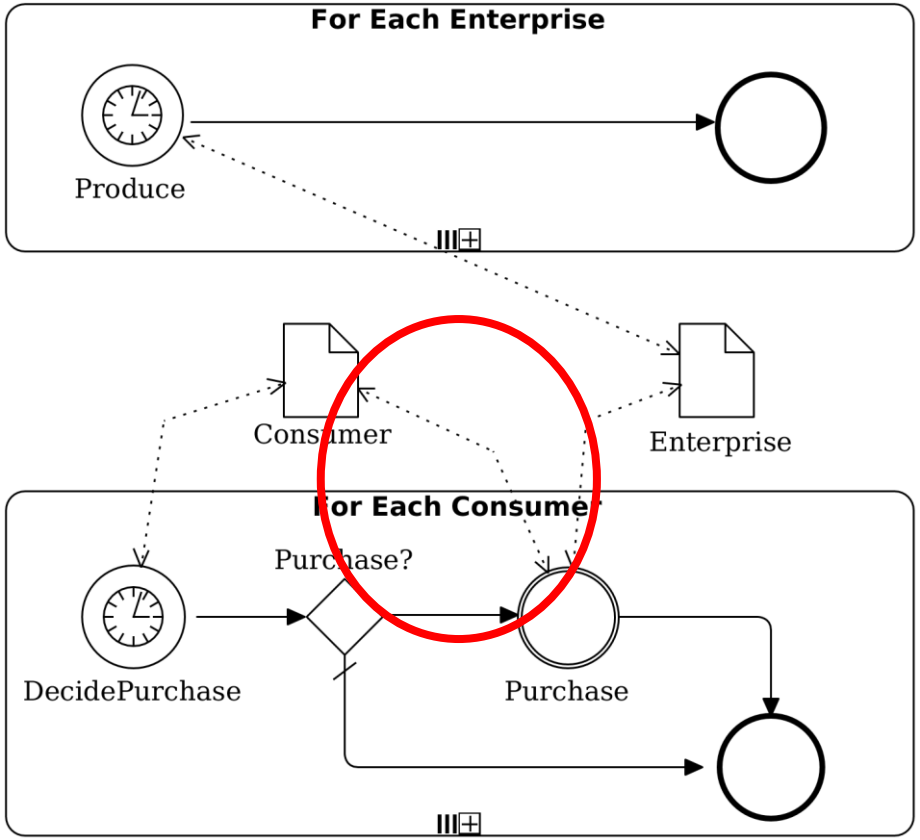
- **Multiple Enterprises** produce and sell one single type of item
- **Multiple Consumers** purchase items from a preferred set of Enterprises
- On each day
  - Enterprises **generate a Produce event** to produce a finite quantity of items
  - Consumers **generates a DecidePurchase event** that generates a **Purchase event** with a **probability** to order a specific quantity of items from **the Enterprise with lowest price in its preferred list**
    - If the ordered quantity is in stock, the Enterprise delivers the quantity items to the Consumer and the order is fulfilled
    - Otherwise, the Enterprise delivers the quantity of items in stock to the Consumer (i.e., order is partially fulfilled), the Enterprise registers the non-fulfilled quantity as lost sales, **and the Consumer replace the Enterprise from its preferred list by another Enterprise**

Purchasing Model – Version 4

https://gnardin.github.io/PurchasingModel/04.html



Class diagram



BPMN diagram

Consumer Object Type

```
var Customer = new cLASS( {
  Name: "Customer",
  shortLabel: "Customer",
  supertypeName: "oBJECT",

  properties: {
    "purchaseProb": { range: "PositiveDecimal", label: "Purchase Prob" },
    "purchaseMin": { range: "NonNegativeInteger", label: "Min Items Purchase" },
    "purchaseMax": { range: "NonNegativeInteger", label: "Max Items Purchase" },
    "purchaseFrom": { range: "Enterprise", minCard: 0, maxCard: Infinity }
  },
  methods: {
    "decideEnterprise": function () {
      var i, e, enterprise, minPrice = Infinity;

      // Select the Enterprise selling the cheapest
      for ( i = 0; i < this.purchaseFrom.length; i += 1 ) {
        e = this.purchaseFrom[ i ];

        if ( minPrice > e.itemPrice ) {
          minPrice = e.itemPrice;
          enterprise = e;
        }
      }
      return enterprise;
    }, ...
```

Output Statistics (simulation.js)

```
sim.model.statistics = {
  ...
  "percSales": {
    range: "Decimal",
    label: "% Sales",
    initialValue: 0,
    showTimeSeries: true,
    computeOnlyAtEnd: false,
    expression: function () {
      var totalOrdered = 0;
      var totalDelivered = 0;
      var enterprises = cLASS[ "Enterprise" ].instances;
      Object.keys( enterprises ).forEach( function ( objId ) {
        totalOrdered += enterprises[ objId ].nmrItemsOrdered;
        totalDelivered += enterprises[ objId ].nmrItemsDelivered;
      } );

      return ( totalDelivered / totalOrdered ) * 100;
    }
  },
};
```

Purchasing Model – Version 4

https://gnardin.github.io/PurchasingModel/04.html

Purchasing Model – Version 4

https://gnardin.github.io/PurchasingModel/04.html

Created with the *Object Event Simulation (OES)* framework ***OESjs*** available from sim4edu.com.

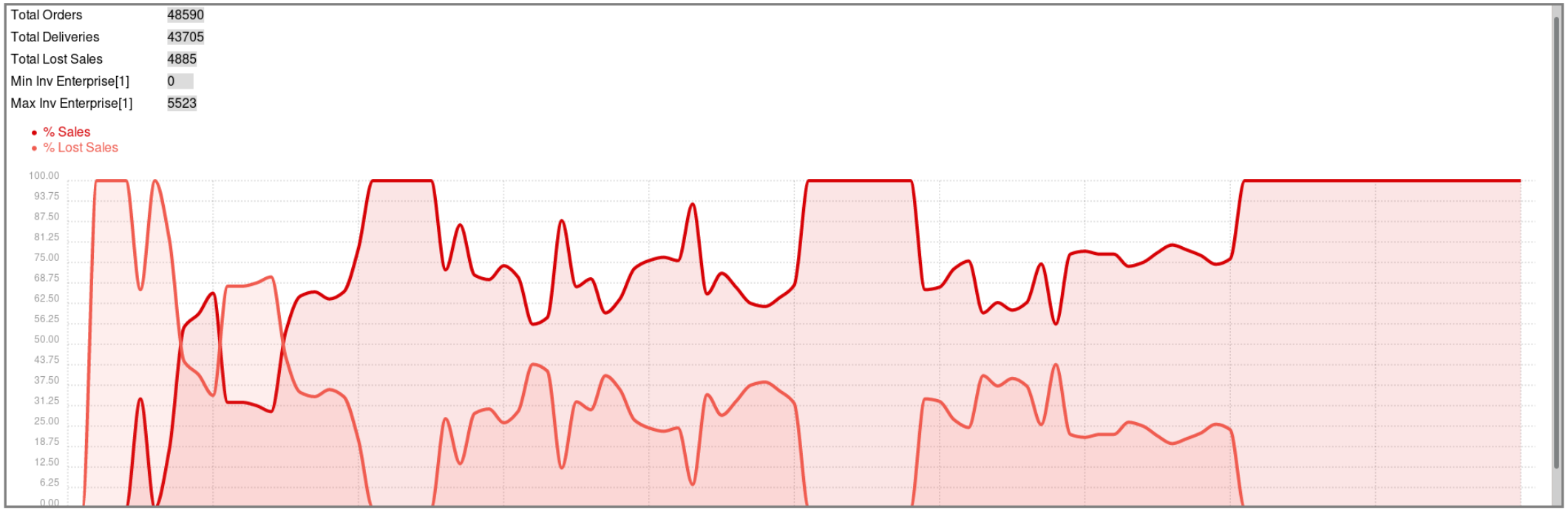
Purchasing Model (4) Read more...

Narrative | Description | Code

Step: 100 Time: 100 D

Stop | Continue | Restart

Total Orders	48590
Total Deliveries	43705
Total Lost Sales	4885
Min Inv Enterprise[1]	0
Max Inv Enterprise[1]	5523



References

- Byrne, J., Heavey, C., & Byrne, P. J. (2010). A review of web-based simulation and supporting tools. *Simulation Modelling Practice and Theory*, 18, 253-276. DOI: 10.1016/jsimpat.2009.09.013.

Thank You!

Appendix

Object-Event Modeling

Object-Event Modeling

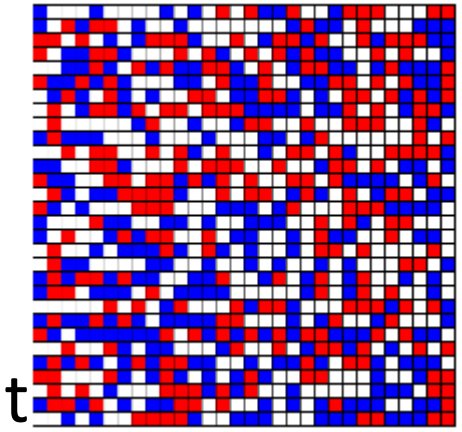
- Any **Discrete Event Simulation** (DES) model has to describe a state transition system in some form
- Object-Event Modeling (OEM) represents a general DES approach based on **object-oriented modeling** and **event scheduling**
- OEM allows the description of discrete event system as a state transition system in terms of
  - **object types**, e.g., in the form of classes of an object-oriented language
  - **event types**, e.g., in the form of classes of an object-oriented language
  - **causal regularities** (disposition types) e.g., in the form of event rules

Object-Event Modeling

OEM is formed of different models

- **Conceptual information modeling**: describe the relevant entity types of a domain and the relationships between them
- **Information design modeling**: describe the platform-independent data structures providing a logical design of a system
- **Data/class modeling**: describe the platform-specific data structures for implementing a system

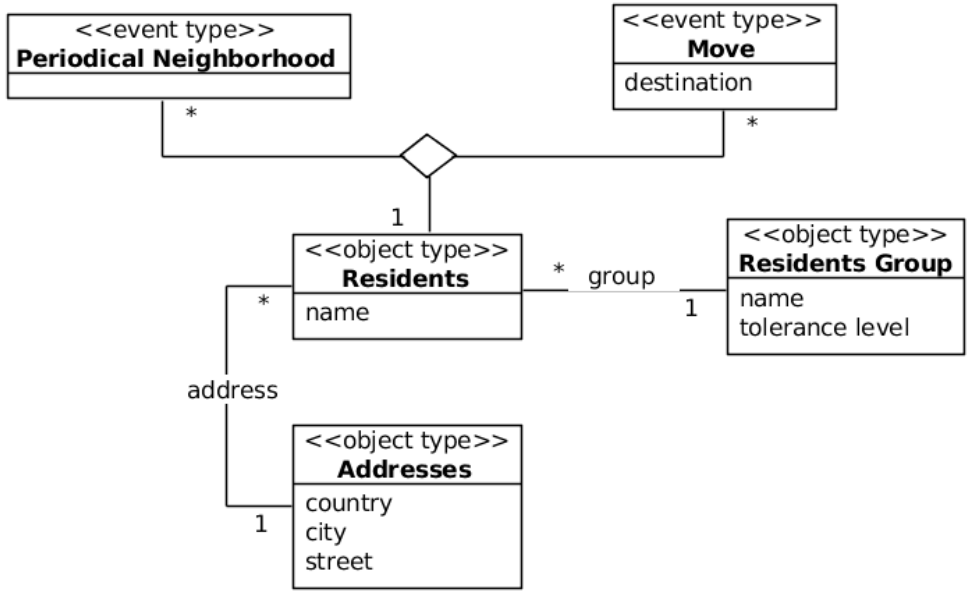
Schelling Segregation

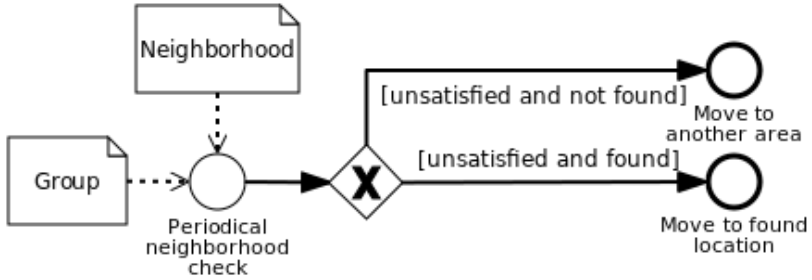


- Residential segregation results from the behavior of resident members of some group, being either satisfied or unsatisfied with their neighborhood depending on the number of residents of the same group in the neighborhood
- Periodically, all residents check if they are content with their neighborhood, based on their degree of tolerating neighbors of a different group
- If they are not satisfied, they move to a location where they are satisfied, or leave the area if they don't find such a location

Conceptual Modeling

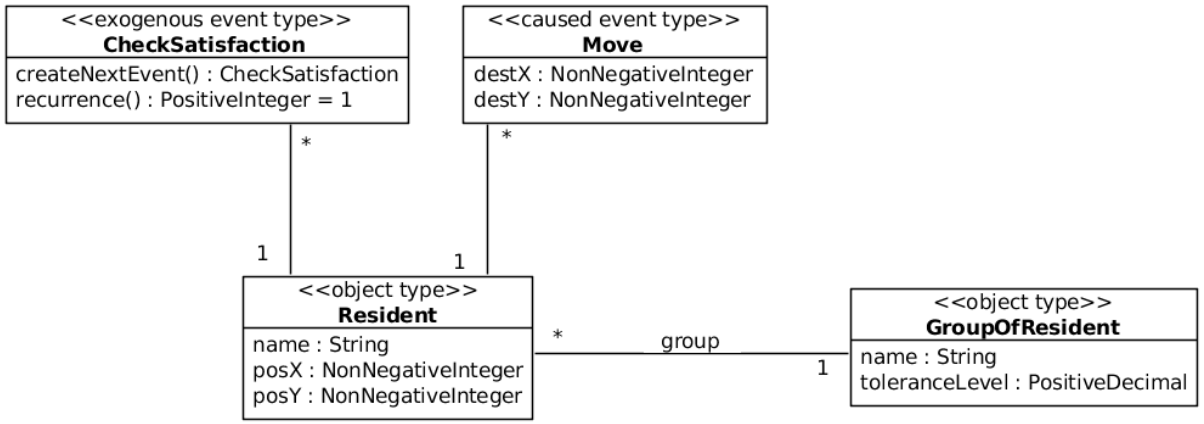
- Identify the Object Types
- Model object as stereotyped classes in a UML class diagram
- Add associations among object types
- Add associations between object types and event types whenever objects (of some type) participate in events (of some type)
- Model event using Business Process Management Notation





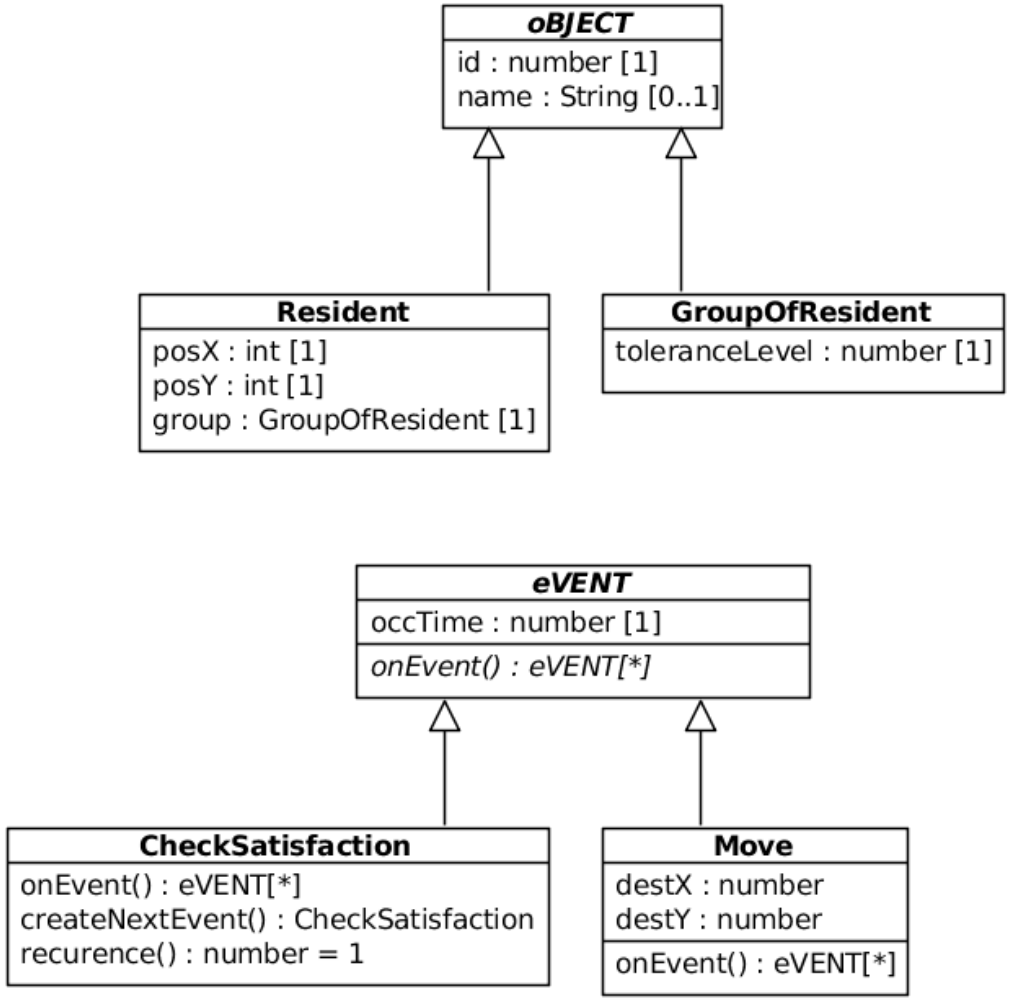
Design Modeling

- Describe the platform-independent data structures providing a logical design of a system
- Simplify the model to answer to a specific research question
- Include properties and functions to the object types and event types



Data/Class Modeling

- Derive platform-specific class models from the Design Model
- Use data types specific to the implementation platform
- Associate the object types and event types to the Abstract classes in the OESjs



Object-Event Modeling

1. Model object and event types as stereotyped classes in a UML class diagram
2. Add associations among object types
3. Add associations between object types and event types whenever objects (of some type) participate in events (of some type)
4. Model random variables as stereotyped operations constrained to implement a certain probability distribution
5. Model event rules in an event rule table associating a triggering event expression with an event routine specified in pseudo-code
6. Model each event rule from the event rule table in the form of an "atomic" BPMN process diagram
7. Model the sequencing of events by merging all "atomic" BPMN process diagrams into one BPMN process diagram, if possible

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

JavaScript Introduction

Introduction to JavaScript

- JavaScript was developed in May 1995 by *Brendan Eich*
- JavaScript is a **scripting language**
  - Lightweight programming language
  - Programming code is embedded in HTML
  - Can be executed by all modern web browsers
- JavaScript is **weakly typed** and **dynamic**
- JavaScript is **object-oriented**, but in a *different* way than *classical OO languages*, e.g.,
  - objects can be created directly without instantiating any class
  - properties can be added to an object or class definition at run-time

Introduction to JavaScript

- JavaScript is **case-sensitive**. Keywords are in lowercase
- Program statements are terminated by **semicolons**
- Curly brackets {…} are used to define **statement blocks**
- Program text between `/*` and `*/` or after `//` is treated as a **comment**
- Variables should be declared at the start of a function using the `var` keyword
- There are only two kinds of scope for variables: **global scope** and **function scope**
- = is used for assigning values to variables
- For testing the equality (or inequality) of two primitive data values, use the strict equality predicate === (and !==) instead of the == (and !=)

`2 == "2"` (output `true`) `2 === "2"` (output `false`)

Introduction to JavaScript

- JavaScript has only three primitive data types
  - ✓ `String`

```
var single = 'Just single quotes';
var double = "Just double quotes";
```

  - ✓ `Number`

```
var num = 10;
var decimal = 19.8;
```

  - ✓ `Boolean`

```
var stop = false;
var active = true;
```

- To test the type of a variable holds `typeof(…)` function

```
typeof(num)
```

(output `number`)

Introduction to JavaScript

- JavaScript has essentially three reference types
  - ✓ `Object` is a set of **property-value-pairs**

```
var person1 = {
  lastName: “Smith”,
  firstName: “John”
}
```

  - Properties can be accessed in two ways:
    - Using the dot notation: `person1.lastName = "Smith"`
    - Using an "associative array" notation: `person1["lastName"] = "Smith“`
  - Looping over the keys of an object:

```
Object.keys( person1 ).forEach( function ( objId) { ... } );
```

Introduction to JavaScript

- JavaScript has essentially three reference types
  - ✓ `Array`

    ```
    var a1 = [1, 2, "Name", true]
    ```

    - Index starts at 0

      `a1[2] = "Name"`
    - Array has a length property

      `a1.length` (output `4`)
    - Arrays can grow dynamically

      `a1[6] = 7`

Introduction to JavaScript

- JavaScript has essentially three reference types
  - ✓ `Function`

```
var myF = function theNameOfMyF () {...}
```

    - procedures are called “functions”, no matter if they return or not a value
    - functions can be stored in a variable
    - can be passed as arguments to functions
    - can be returned by functions
    - function name can be omitted (e.g., omit `theNameOfMyF`), then the function can be invoked using the variable it is assigned to, like `myF()`

Other Model Configuration

simulation.html

- Standard file that launches the simulation model, requires small changes
  - Tab title

```
<head>
  <meta charset="utf-8"/>
  <title>Purchasing Model (1)</title>
```

  - Front title

```
<div id="frontMatter">
  <div id="sim4eduinfo">Created with the <i>Object Event Simulation
(OES)</i> framework <b><i>OESjs</i></b> available from
    <a href='https://sim4edu.com'>sim4edu.com</a>.</div>
  <h1>Purchasing Model (1)</h1>
</div>
```

description.html

- Standard file that contains the description of the simulation model
  - Tab title

```
<head>
  <meta charset="utf-8"/>
  <title>Purchasing Model (1)</title>
```

  - Description

```
<body onload="oes.ui.setupDescription()">
  <div id="frontMatter">
    <h1>Purchasing Model</h1>
    <p><strong>Classification tags</strong>: business operations management, DES, next-event
time progression</p>
    <section id="shortDescription">
    </section>
  </div>
        ...
</body>
```

Simulation Model Configuration (simulation.js)

Property	Description
`sim.config.stepDuration`	Time in `ms` between timesteps
`sim.config.createLog`	Enable/Disable the simulator log on the screen
`sim.config.visualize`	Enable/Disable the visualization of the simulation execution (Note: Enable only if the model has visualization)
`sim.config.userInteractive`	Enable/Disable the user interactivity feature (Note: Enable only if the model has user interactivity)

Global Functions (simulation.js)

```
/* Global Functions */
sim.model.f.produceItems = function( e ) {
  e.inventoryLevel += rand.uniformInt( e.productionRateMin, e.productionRateMax);
};
```

- Because functions do not change during the simulation execution, to call a global function use the `sim.model.f.<name>`

Space Configuration (simulation.js)

- OESjs allows to represent the space as a 2D-Grid
- The grid dimensions are defined by the `sim.model.space` object

Property	Description
`Sim.model.space.type`	Define the type of space to be represented (`IntegerGrid`)
`sim.model.space.xMax`	Width of the grid
`sim.model.space.yMax`	Height of the grid

```
// Space model
sim.model.space.type = "IntegerGrid";
sim.model.space.xMax = 120;
sim.model.space.yMax = 60;
```

Grid Space Configuration (simulation.js)

- To access the content of the grid cells

```
sim.space.grid[(y-1)*xMax + x-1]
```

```
oes.space.grid.0.getCellValue(x,y)
```

- To loop over all cells of the grid

```
oes.space.grid.forAllCells( function (x,y) {
    var g = oes.space.grid.0.getCellValue(x,y),  tol = 0;

    if (g > 0) {
      tol = groups[String(g)].toleranceLevel;
      if (neighbDiffLevel(x,y,g) > tol) {
        sim.v.uncontentResidents.push([x,y]);
      }
    }
```