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

THE TAO OF SIMULATION

Abhinav Adduri

Computer Science
University of California, Berkeley
Berkeley, CA 94720 USA

Lee Schruben

Industrial Engineering and Operations Research
University of California, Berkeley
Berkeley, CA, 94720 USA

ABSTRACT

This paper introduces an open-source, cross-platform object called Tao to the simulation community for teaching and research. Tao is simple enough to begin using within minutes, but is extensible and interoperable with powerful statistical, graphical, animation, and simulation software, as well as internet-of-things hardware. Tao runs locally or remotely in a web browser, freeing it from OS constraints to integrate simulation engines directly into application-specific websites.

Modeling examples presented here include a chemical reaction, time-bound activity sequences, and an embedded simulation for real-time load balancing. To illustrate using Tao for research, Hyden's, single-run, global optimization algorithm was implemented. This extension enables Tao to optimize the design of dynamic systems such as queueing networks and supply chains in a single run. To demonstrate Tao's extensibility, the innovations of pending-intervals and a bioproduction stochastic scheduling algorithm were implemented.

1 INTRODUCTION

First off: an explanation of the pretentious-sounding title for this paper is in order. The Tao project was originally named Tau, envisioned years ago simply as a web-based, open-source replacement for Sigma, (simulation educational software developed by the second author in the 1980's). Sigma was created back in the age of acronyms (it stands for SImulation Graphical Modeling and Analysis), and Tau is simply the next letter in the Greek alphabet. "Tau" eventually morphed into "Tao" as the seemingly limitless possibilities for simulation modeling on the internet became apparent. This first release was delayed to investigate numerous possible extension or applications. So, with apologies to those whose sensibilities it might offend: Tao now seems an apt name for this "journey that never ends"; and it already has a logo, ☯.

Tao has its roots in over three decades of using Sigma, the first graphical discrete-event simulation software to run in the DOS operating system (Sigma predates the mouse). Sigma was rewritten to become the first simulation software to run under native 16-bit MS Windows. Sigma was offered to the simulation community as a teaching and research resource and never intended for developing commercial simulation software. However, it eventually moved in that direction. Tao source code is likewise freely offered and, hopefully, not destined for commercialization. Over the years, Sigma has been used by thousands of students in the classroom, in simulation research and graduate theses, and ultimately to generate fast, flexible simulation engines for commercial application-specific software. Videos for a wide variety of simulation "term products" created with Sigma can be found by searching YouTube with {simulation Berkeley IEOR131}. Tao has learned from mistakes in designing Sigma, most notably an obsession with minimalism; giving users access to full modeling power, and full responsibility for their models (programming required!). While doing much more for its users than Sigma, Tao has still tried to avoid what Pidd and Carvalho call commercial simulation package "bloat" (Pidd and Carvalho, 2006),

primarily so simulation students can focus on modeling fundamentals and researchers can test new ideas without being constrained by a particular Weltanschauung.

Sigma for Windows was created when sophisticated tricks were required to code the dynamic overlays needed to execute within a 640K. Open-source software development at a time when almost nobody knew how to program, or even use, computer graphics was pointless. Tao, however, is written in JavaScript, with a huge developer base and is used on almost every website.

Even after commercial simulation software was released that ran under 32-bit MS Windows, Sigma remained surprisingly robust and durable (it won an EDUCOM software award almost a decade after it was created (Boettcher, 1993)). It has been only slightly modified over decades of use, mostly to accommodate changes in Microsoft Windows.

Simply making Sigma independent of MS Windows was the original goal of the Tao (née Tau) project. However, modern internet technology has allowed a wealth of useful and innovative new features to be included. Tao has modeling power and flexibility not yet freely available to the simulation community. Throughout the years of using Sigma, many practical applications where found where it's basic event graph paradigm was cumbersome. Sigma was also difficult to use in embedded simulations since there is no native communication enabling it to talk to external devices and services (although it generated portable C code with simlib functionality (Law, 2014)). Tao was developed to overcome these shortcomings and was designed to provide an efficient platform for simulation modeling anywhere.

Tao runs locally or as a client in any JavaScript-enabled web browser. This makes effective open-source code development possible. Tao itself employs some advanced JavaScript features. However, the JavaScript required to ***use*** Tao to build any discrete-event simulation model can be acquired quickly. In the classroom, a single-sheet introduction to JavaScript has been sufficient for students to begin using Tao. This is reproduced in the Appendix.

Tao is extensible and interoperable with a vast array of powerful statistical analysis and animation software. It is both interactive and fast (using native JIT compiling). This makes Tao an easily accessible platform for learning about simulation, and flexible tool for simulation research. Nelson lamented in his recent textbook that commercial simulation languages did not provide a flexible enough platform for research (Nelson, 2013). Tao is designed in part to remedy that longstanding problem. It also is designed to provide a fun context to generate interest in learning more about JavaScript and explore how simulations can be integrated with internet-of-things hardware.

2 TAO

Users need a computer (any OS), an internet connection, and a web browser with JavaScript enabled. Launch Tao by simply navigating to *tao-simulation.herokuapp.com* (everyone should take security precautions whenever using the internet, and do so it their own risk). Once all the Tao simulation assets are loaded, one can begin creating any discrete-event simulation model in the simulation playground. The core discrete-event simulation object is the `scheduler`; to find the simulated time use `scheduler.getClock()`. This object can be viewed or modified in the source code to explore different event list processing algorithms.

The most basic use of Tao is for creating enriched event graph simulations: double clicking the simulation playground creates an event vertex for defining system state changes, and [shift-clicking] connects two event vertices with directed edges that specify the relationships between events. Two basic event-scheduling and event-canceling edges are all that are necessary to give Tao the full modeling power of a Sigma and a Turing machine (https://en.wikipedia.org/wiki/Church-Turing_thesis) (Savage, et. al. 2005). Several other innovative and useful constructs have been added to illustrate Tao's extensibility.

2.1 The Tao Simulation Playground

Use the up/down arrow keys (or touch screen) to navigate and [Ctl +] and Ctl -] to zoom in and out as in any conventional web site. To keep the layout simple, web panels are used. Personalization can be added to one's own playground as in any web site.

2.2 Event Objects

In order to create an "event" object for any JavaScript program, double click anywhere in the gridded playground. An event vertex will be created at your cursor's position. To edit an event, simply click once on it. This will open an Event Editing Panel to edit the following features of an event:

- Name: This is the event object name used internally and displayed.
- Description: This tells what you intend will happen whenever this event occurs.
- Trace: This field determines if this event will be observed for debugging or analysis when the simulation is run. A checked box means event execution will be recorded; uncheck it to hide its execution (this useful for re-using generic background objects such as IoT nodes)
- State Changes: The state changes box contains any JavaScript code. This code will be run when the simulated global time reaches the scheduled event execution time – or whenever a condition it is watching occurs. Within the state change, one can access global variables, parameters, functions, and objects or create private variables, functions, or objects.
- Parameters: Each event has optional input parameter values as arguments. To add parameters, put the desired argument name in the input box and press enter. Each edge event can pass its own values to an event object as input arguments. To access a parameter within a state change, use the `params` object. For example, a parameter named x would be accessed with the customary object dot notation (`params.x`).
- Private Variables: Tao allows for private variables, functions, and objects – visible only to a particular event object and its exiting edge objects during execution. For example: to use a private variable in an event, just initialize it with proper JavaScript syntax in the state change box, i.e. `var x = 5;` .

2.3 Edges: Event Relationships

To model conditional and dynamic causal relationships between events, an directional edge between them is created. To create an edge in Tao, hold down the 'Shift' key and click on the source event, and then click on the target event. To edit an edge, click on its arrowhead. This opens an Edge Editing Panel. where various attributes about any edge can be modified. Attributes can involve global variables, and both parameters and private variables of an edge's source event. The following are basic edge fields

Edge Type: There are currently several edge types in Tao, selected in a drop-down menu.

- Scheduling Edges: This is the fundamental causal relationship between system events and underpins the logic of all discrete-event simulation software, and the only edge necessary for a full modeling power (Turing-Complete) modeling tool.

Condition: This is any Boolean expression. The condition of an edge governs its existence; while an edge condition is FALSE (or 0) it simply does not exist. An edge is enabled only while its condition evaluates to the Boolean value TRUE (or 1).

Delay: An edge delay governs how long (usually random) until an event is executed after its condition tests as TRUE. The edge delay time code evaluates to any numerical value, allowing negative times (there are several practical reasons for looking backwards in time, e.g., for modeling recourse)

Priority: The priority field is any numerically-valued expression. This allows dynamic control of which event executes next if there were otherwise a collision. Note events have execution priorities entirely dependent on how they were scheduled, not what an event does – this was a critically important lesson from developing Sigma and naturally avoids some nasty (hidden) simulation behavior such as simulation resource deadlock (Venkatesh, et. al. 1998)

- Pending Edges: A pending edge watches for an edge enabling condition to become TRUE. The edge quits watching when a terminating condition becomes TRUE; for example, a worker looks for work only during the remainder of their shift. This is a variation on the powerful event graph extension embodied in LEGOS (Buss and Sanchez, 2002).

- Cancelling Edges: Cancelling edges allow a user to cancel an event based on a condition after a given delay. It can cancel the first occurrence, a specific instance, or all instances of an event after the specified delay, if any, has passed.

- Predecessor and Successor Edges: These are innovative Tao extensions created to illustrate Tao's extensibility.

2.4 Parameters

Parameters are essential for simulating very large systems with very small models. They allow events to pass information to other events (such as entities in a process flow) that can possibly alter the simulation rules of execution. A parameter's value can be a global variable, a private variable, or a source event parameter variable.

2.5 Global Variables

To create a global variable – such as a system key performance indicator (KPI), click on the 'Globals' tab on the right side of the screen. Go to the input under the 'Global Variables' label, type in the name of the variable, and press enter. A new global variable with an initial value of zero will be created. You can then edit the value and description of the global variable. To delete a global variable, press the 'x' button next to the corresponding variable. Global variables are accessible by any edge or event using the '`globals`' object. For example, a global variable named `servers` could be incremented by one as follows: `globals.servers+=1;`.

2.6 Run Control

Tao can ***simultaneously*** run different simulation models together in parallel execution "contexts". Thus competing designs can "race" instead of running sequential "time trials" like typical run-based experiments. Each context can have different configurations (experimental design points) specified for its

global variables. This powerful feature enables, for example, execution of multiple replications of an entire experimental design in a single run, or embedding a self-optimizing control algorithm into a model. To change the number of contexts, change the number in the input field labeled "Contexts". Then, list each global input variables in a comma-separated array specifying initial values to the respective contexts. For example, a simulation with global variables 'servers' and 'queue' with two contexts could have values of [1,2] and [3,3] respectively. For context 1, servers and queue are initialized to 1 and 3 respectively, and for context 2, servers and queue are initialized to 2 and 3 respectively.

As an example of using "context", Hyden's single-run global simulation "Time Dilation" algorithm has been implemented in Tao. This enables Tao to select and batch-replicate the optimal system automatically in a single run *without knowing in advance which is best*, thus integrating exploration and exploitation for simulation optimization (Hyden, 2002).

Each simulation is run for a certain amount of time or general terminating condition. A simulation run will also terminate whenever there are no further scheduled or pending events.

After a simulation is run, a full event trace is available for debugging by opening the JavaScript web browser console: ([command + shift + j] on a mac, or [ctrl + shift + j] on a Windows pc). Tao graphs global available by clicking the "Graphs" button. There is an option to change plot types. Developers can insert various output graphics or animations from the rich array of open-source JavaScript codes.

2.7 Reusing and Sharing Models

To save or re-load a model, click the 'Download Tao File' button at the bottom of the 'Globals' panel.

2.8 Generating Portable Simulation Engines

Tao allows a user to download the required JavaScript code to execute any simulation engine elsewhere such as locally as part of other software (like an optimization oracle) or embedded in hardware or a website. To do this, software that can run JavaScript is required. To download such software, visit https://nodejs.org/en/. This also enables access to the internet of things for integrating a simulation with hardware.

3 TAO INNOVATIONS AND INTERNAL LOGIC

Tao is a developed for the web browser to facilitate cross-platform dynamic simulation. Every event object in Tao is itself Turing-complete, which is an advantage over the simple state change per event execution model introduced in Sigma, and underpinning most simulation software execution. A user is able to define private variables, functions, and do anything else normal JavaScript allows inside the body of any event. Any necessarily persistent variables or functions may then be passed via parameters to other events, allowing multiple events to have access to any particular object. Parameters for any event are given the value computed in each incoming edge, and the value itself is interpreted during the insertion of the target event into the future events list. This allows for private variables defined in the source event, as well as any helper functions, to be passed into the target event. This lazy evaluation of parameter values allows a user to control the scheduling behavior of the target event.

Tao inherits a wealth of functionality from JavaScript, such as live data feeds, that a user can use in any part of a simulation model, including inside events or in edge time delays. Examples include random variate generators, derivatives and integrals, and other mathematical, statistical, graphical, animation, hardware code, and Runge-Kutta algorithms for continuous time simulations.

Event relationships in Tao are significantly more sophisticated than they are in the minimalist event-graph methodology. There are several privileged functions available to the user to take finer control over the scheduling system in Tao. The user can get and reset the current clock time in the state change of any

event (or edge, since edge delay times and conditions are evaluated at run time). The user can request information about the total number of events processed so far, get the number of scheduled events of a particular type, or get the number of cancelled events of a particular type. The user can request to terminate the simulation as well. The scheduling system allows the user to trap the simulation to the scheduler, meaning a user can adjust any scheduler parameters necessary by passing in an anonymous function to the trap function. Lastly, an advanced feature of the scheduling system allows a user to send out synchronous HTTP compliant requests to a target URL. This allows a simulation to get and post data to a remote server, or embedded system in real time, enabling Tao simulations to have fine control over external electronics or processes.

One possible issue with developing a simulation engine in a browser is limitations set by the browser on how much processing power a single tab can utilize. To help alleviate this problem for computationally intensive models, Tao allows users to download a portable Javascript executable engine of their simulation model. This executable can then be run locally in the terminal, and the output of the simulation can be traced in the terminal console. External intervention in the form of interrupts were necessary because the browser does not allow for graphical interaction during heavy computational tasks. For this reason, if the user unintentionally causes an infinite loop (for example, by looping over an edge with a delay of zero), there is no way to allow the user to terminate a simulation. To avoid this problem, if a certain number of events have been scheduled without any increment in global time (this limit is initialized to one hundred time units, but can be set by the user), the user is prompted to continue or terminate the simulation.

Every simulation in Tao can be run with concurrently executing models. Since JavaScript is inherently a single-threaded language, these different models are multiplexed in time. These "contexts" are provided with different initial values in the graphical interface, and are completely separate from one another once launched. There is no communication between models in different Tao contexts as currently implemented, and as such there is no concept of a lock over shared resources. These could be added to allow algorithms such as Agent-based Particle Swarm Optimizations (where each "particle" is a complete simulation model).

4 ILLUSTRATIVE EXAMPLES

The hundreds of models included with the basic Sigma download (from www.sigmwiki.com) can be easily simulated with Tao, and optimized if desired. Some more unusual discrete-event simulation examples are illustrated next.

4.1 Hydrazine Production

This simulation of a chemical reaction was created to illustrate the use of pending edges. The discrete events are molecular collisions. Hydrazine is an important, highly-unstable compound used in many products including rocket fuel, pharmaceuticals, and oxygen scavengers in both nuclear and conventional power plants. In Figure 1 a pending edge from Hydrazine to Reverse models how the decomposition of hydrazine is conditional on its concentration. The delay and condition associated with this edge can be modified to match the empirical stability of Hydrazine, and various extraneous factors such as temperature and pressure can be factored in through global variables or parameters if necessary. Since Tao allows for edge delays to be calculated during the resolution of the source event, it allows for fine grain control over the exact rates of decomposition of Hydrazine. The concentration dependent reaction is representative of chemical equilibrium: for example, by increasing the starting concentration of Hydrogen and Nitrogen, the probability that a molecule of Hydrazine is formed increases, mimicking the natural Le Chatelier equilibrium process. The Hydrazine event can also be modeled as an intermediate step towards a multi-step synthesis by adding an outgoing pending edge from Hydrazine to the next step.

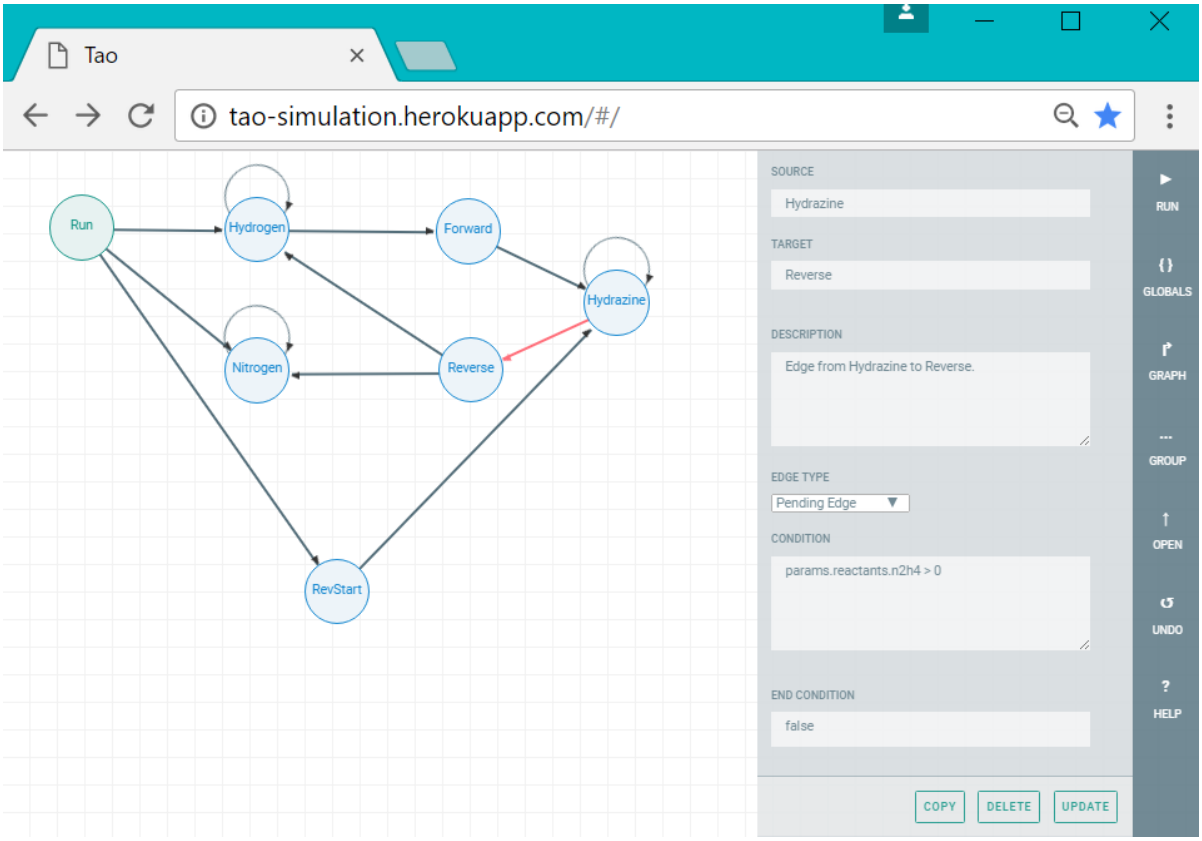

Figure 1: Hydrazine production – with concentration dependent reactions

4.2 Scheduling Time-critical Activity Sequences

Consider simulation-based scheduling which is modeling a process that at a certain critical juncture there are several antecedent activities that must take place before that critical activity can begin, but not too far in advance. Such time-bound sequences are common in hi-tech manufacturing, communication, medical procedures, and construction (e.g., paint primer can't be too wet or too dry before applying the next coat).

There are two ways time-bound activities can be implemented using the Tao toolset. To illustrate: assume there are three antecedent events (A, B and C) that must proceed some critical event (D) within time-interval tolerances. In the first approach, the user can create a variable to keep track of how many times the events A, B, and C have been run. The user can create a pending edge directly to the critical event D, with the condition that the number of times A, B, and C have been executed is one more than the current amount (stored in the aforementioned variable). The user would then increment the number of times A, B, and C have been executed in that variable, and the simulation would proceed through the critical juncture.

A cleaner approach involves relative-timing of activities and is a powerful generalization to Activity Networks (such as PERT charts) that greatly extends their flexibility, utility, and ultimate robustness. Such networks are easily simulated in Tao using predecessor and successor edges, which may be discussed in the presentation if time allows. Such an approach would still require the use of pending edges to minimize wait time between the last step of an antecedent critical juncture and the first step of a following critical juncture.

4.3 Embedded Simulation for Communication Load Balancing

Consider the case of a server farm that processes a large incoming number of requests, necessitating the use of a load balancer. Assume that each computer that the company uses costs money, and so the load balancer only requests access to another computer if latency is likely to exceed an acceptable threshold. Conversely, the load balancer can take computers off-line if they are not expected to be needed in the near-term future. Using simulation, the company could track the incoming server requests over the past few seconds to create a gradient model and spin up the minimum number of servers needed to facilitate fast service. The delay times in response could be taken in as HTTP requests by the Tao model, allowing for a simple embedded system to keep track of how many servers the load balancer should make activate of at given time. The simulation model would serve as a real-time solution to the otherwise complicated stochastic scheduling problem. In order to transmit this information back, the Tao model can emit HTTP responses containing the calculated number of servers. These HTTP requests can be made to be either synchronous or asynchronous. There are merits to each method: synchronous Tao modeling simplifies logic flow but is generally not suited to handle a large number of small requests; asynchronous programming is more complicated but allows the application to reduce request wait times.

5 ACKNOWLEDGMENTS

While many people contributed code to Sigma, two were essential. The second author knows only one living software developer who thrived under the early development constraints for Sigma, Quint King; the only other person with these skills he ever knew was Kevin Healy. The Tau Project was initiated by Pranava Adduri 3 years ago in his Master's of Science thesis. The authors thank the Bioproduction Group (Bio-G.com) for their challenging problems and support.

6 APPENDIX: JAVASCRIPT BASICS

6.1 Preliminaries

This brief section introduces all the JavaScript that is needed to create robust, complete Tao simulation models. Only a cursory knowledge of JavaScript needed to *use* Tao (Tao code itself is more advanced.) This frees classroom time from particular software syntax details for exploring deeper aspects of the art of simulation modeling and analysis. There are many good on-line JavaScript tutorials Presented here are the simply the minimal basics to begin using Tao effectively.

JavaScript is a dynamic, object-oriented programming language. It has several similarities to other programming languages, but uses some unique advanced programming concepts. In the following code segments, non-executing comments follow the // on each line.

6.2 Variables

JavaScript allows variables that are not strongly typed. For example, the following statements:

```
var str = "A sample string.";
console.log(str);
str = 4;
console.log(str);
```

will print "A sample string" and "4" respectively. Integers and floating point variables can be manipulated as in any programming language. Variables are passed by value, and objects passed by reference.

6.3 Functions

Functions are objects in JavaScript. It is possible to assign a function to a variable, as well as use functions as arguments to other functions. For example, the following two sets of code do the same thing.

```
var foo = function(x) { return x*x; }
var execute = function(func, args) {
  return func(args);
}
execute(foo, 5);
```

```
function square(x) { return x * x; }
function execute(func, args) {
  return func(args);
}
execute(square, 5);
```

6.4 Objects

Functions can describe objects which can be instantiated using the "new" keyword. But for conventional simulation modeling, it is unusual to need to create objects in this manner. It is possible to create an object and store its reference in a much simpler fashion. For example, the following creates an object and stores it in a variable called car. (Note the use of quotes to define strings.)

```
function Car(brand, seats, color) {
 return  {
   'brand': brand,
   'seats': seats,
   'color': color,
   'print': function() {
      console.log(this.brand, this.seats, this.color);
   }
 };
}

var car = Car('Honda', 4, 'Red');
car.print(); // prints "Honda 4 Red"
console.log(car.brand, car.seats, car.color); // also prints "Honda 4 Red"
```

It should also be noted that this is not strictly a properly designed object. But, such entity structures suffice for most simulation models. Also, there is no need to use a function to return an object in the above manner. It is possible to directly assign a variable to an object, and dynamically access an object's properties using such variables. To illustrate:

```
var car = {'brand': 'Honda', 'seats': 4, 'color': 'Red'};
console.log(car.brand, car.seats. car.color); // prints "Honda 4 Red"

var property = 'brand';
console.log(car[property]); // prints "Honda"
```

Every time one uses the {} syntax to make an object, a new object is created. For example:

```
for (var i = 0; i < 5; i++) {
  var person = {'age': i};
}

console.log(person.age) // prints out 4
```

The price for this simplicity is the references to the first four objects are lost in the above code. Lastly, you can also dynamically add properties to objects.

```
var carwash = {};
carwash.servers = 5; // same as carwash['servers'] = 5
carwash.queue = 3; // same as carwash['queue'] = 3
```

7 REFERENCES

Boettcher, J. V. 1993. 101 success stories of information technology in higher education: The Joe Wyatt challenge. New York: McGraw-Hill.

Buss, A., and Paul J. Sánchez, (2002) "Building Complex Models with LEGOs (Listener Event Graph Objects)", *Proc. 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, Institute of Electrical and Electronics Engineers, pp. 732-737.

Hyden, P. (2002) "Time dilation: Decreasing time to decision with discrete-event simulation", PhD Thesis, IEOR Department, Cornell University.

Law. A. M. (2014), Simulation Modeling and Analysis, 5th ed., McGraw-Hill.

Nelson, B. L. (2013) Foundations and Methods of Stochastic Simulation: A First Course, Springer-Verlag.

Pidd, M., and A. Carvalho, (2006), "Simulation software: not the same yesterday, today or forever," Journal of Simulation, 1.1: 7-20.

Savage, E. L., L. Schruben, and E. Yücesan (2005), "On the Generality of Event-Graph Models" INFORMS Journal on Computing 17.1: 3–9.

Venkatesh, S., J Smith, B L Deuermeyer, G L Curry, (1998),"Deadlock detection and resolution for discrete event simulation: Multiple-unit seizes" IIE Transactions, 30.3: 201–216.

AUTHOR BIOGRAPHIES

ABHINAV ADDURI is a CS major at the University of California, Berkeley. His e-mail address is abhinadduri@gmail.com.

LEE SCHRUBEN is on the Faculty at the University of California, Berkeley. His e-mail address is LeeS@Berkeley.edu