Zero-JavaScript Stateful User Interface Framework

Matthew Steven Frisbie
Zero-JavaScript Stateful User Interface Framework

The JavaScript language was created so that web pages could become dynamic. JavaScript allows for nearly unlimited extensibility in terms of making web pages do pretty much anything a developer can think of. JavaScript is an interpreted language, which means that code sent to the browser is executed as the code is delivered to the browser (instead of being compiled beforehand). This may be slow, and may cause web pages to be bloated because developers included too much JavaScript. This document discusses a solution to provide some of the functionality of JavaScript without using JavaScript.

Without using JavaScript, a developer had difficulty creating a piece of a web page to maintain state, for example, the ability of a web page to remember and reference previous behavior and information. State may be useful in the context of a user interface. For example, using HTML and pseudo-JavaScript, a developer may create a button and a box.

```html
<button></button>

<box visible=false></box>
```

The initial state of this system (when the page loads) is that the box is hidden. The developer may wish to make this system stateful by making the box visible when the button is clicked. With JavaScript, a developer would add a click listener to the button, and a click handler that executes when a click happens to change the visibility of the box.

```javascript
var button = get('button')

button.onClick(function() {
```
button.visible = true

})

The above example is a stateful system. The example uses a click input by the user to transition from the first and initial state, to a second and final state. For a system to be stateful, there should be some store of information that maintains the state. This information store can take many forms and should be writable. In this example, the writable state is maintained inside the button element, which has a visibility property that may persist through the lifetime of the page. This document discusses a solution to creating a stateful user interface without the use of JavaScript. This solution may be useful in serving advertisements, where the format and content of what an advertisement provider may place on publisher pages may be limited. May other uses existed outside of the advertising space where a developer wishes to have a lightweight, stateful web page without using JavaScript.

To create a stateful web page without using JavaScript, it may be helpful to provide an example definition of stateful. For a user interface to be stateful, the user interface should be able to be represented and act as a “finite state machine.” A finite state machine may have several requirements for a user interface finite state machine.

The state machine should have the ability for the system to assume different states. A finite state machine may only exist in one state at any given time. The state machine should have the ability to recognize what state the system is in, and to project that state onto the user interface. The state machine should have the ability to transition between states based on some outside input. For the purposes of web environments, this may relate to “click events,” which may be individual clicks of a mouse, touches on a screen, voice input, or any other similar input.
A developer should be able to represent a user interface as a “state diagram.” In some instances, the state diagram may not have an output. Instead, the output may be the user interface changing.

Click events may be an input that will change the state of the system, and a developer should have a way for the system to “remember” that a click occurred. Browser click events may be transient. For example, if a user clicks a certain part of a page, and there is no JavaScript set up to detect that a click occurred, the click event will traverse through the page and then be destroyed. Upon the next click, the browser will have no memory that the initial click occurred. This is an example of a stateless system.

HTML does have a construct that can hold state, for example, a form. Forms may be used in various pages throughout the internet for their intended purpose, for example, to collect information in a variety of forms, and to relate all the pieces of together. Forms may have various elements that developers may be familiar with such as text inputs, checkboxes, radio buttons, submit buttons, etc. Many browsers natively support elements.

To create state, the form input elements that may be useful are the radio button and the checkbox. Radio buttons may be defined to be part of a collection and identified by name. In a collection of radio buttons, either no buttons are selected, or only one button can be selected at any given time. Radio buttons are selected by clicking them. Clicking a selected radio button again will have no effect, the radio button will remain selected. Clicking a different radio button will change that radio button to be selected, and the original will be deselected.

Checkboxes in a form may be similar to radio buttons, but there is no enforced exclusive state. Clicking a checkbox will check the box, and clicking it again will uncheck it. Checkboxes
can be assigned to a collection, but clicks on other checkboxes in a collection may have no effect between each other. Each checkbox may be independently toggled by clicking it.

Although both of these allow a developer to maintain state, the example below discusses using radio button input for constructing a finite state machine. A “checked” radio button may represent an active state, and each individual radio button can represent a different state. These require no JavaScript to operate.

By itself, a set of radio buttons may be considered a finite state machine. However, this state machine may not be satisfactory for a developer who wishes to have ability to make our user interface appear in any form. Also, a developer should have the ability for different inputs to cause a transition to one state.

To set an initial state for both radio buttons and checkboxes, each be assigned an initial value when the page loads. This assignment may be done by giving the <input> tag a checked=“checked” attribute. The browser constraints for radio buttons may still be applied. This allows a developer to set an initial state for the system.

For projecting state, a developer should have a form to manage the state of a system. The developer may wish to have the ability to use different HTML to construct the user interface. The developer may use CSS pseudoselectors to project the checked state of a radio button onto HTML elsewhere, and toggle visibility, animations, and other user interface elements.

The ~ selector in CSS is the general sibling selector. When a selector condition is met, it allows a developer to apply a style to any element at the same level in the DOM that matches the defined conditions. The :checked pseudoselector in CSS is a property which is applied when a “checkable” element becomes checked. This can be used for both radio buttons or checkboxes.
With these two CSS tools, a developer may be able to project the state of radio buttons onto sibling HTML, which can act as the user interface.

```
<style>
#view1,
#view2,
#view3 {
  display: none;
}

#state1:checked ~ #view1,
#state2:checked ~ #view2,
#state3:checked ~ #view3 {
  display: block;
}
</style>

/*Click a radio button to change the state*/
<form>
  <input id=state1 type=radio name=collectionA checked=checked>
  <input id=state2 type=radio name=collectionA>
  <input id=state3 type=radio name=collectionA>
</form>
```
The following images illustrate the three different states of this example.

View 1

View 2

View 3

For transitioning state, a developer should have the ability to interact with different pieces of the user interface, instead of interacting with the form. This interaction can be accomplished with the “for” attribute of form labels. This attribute allows a developer to associate a label, which is an HTML element that can be arbitrarily styled, with an input inside the same form (associated by the matching ID). Multiple labels can be associated to the same input. Additionally, clicks on this label may serve as a click on the associated form element.

Therefore, this technique allows a developer to add components inside the views that can take clicks as inputs and mutate the state of the system. These components, or labels, can be styled as buttons, menus, anything that a developer may normally do on a web page.

Furthermore, since these labels can act as the inputs, the original radio buttons can be set to hidden, and the user interface can stand on its own.
<style>
input,
#view1,
#view2,
#view3,
#view4 {
  display: none;
}

div {
  padding: 5px;
  border: 1px solid black;
}

label {
  display: inline-block;
  padding: 5px;
  margin: 5px;
  color: white;
  background-color: black;
}

#state1:checked ~ #view1,
#state2:checked ~ #view2,
#state3:checked ~ #view3,
#state4:checked ~ #view4 {
  display: block;
}
</style>

<form>
  <input id=state1 type=radio name=collectionA checked=checked>
  <input id=state2 type=radio name=collectionA>
  <input id=state3 type=radio name=collectionA>
  <input id=state4 type=radio name=collectionA>

  <div id=view1>
    <h2>View 1</h2>
    <label for=state2>Go to state 2</label>
    <label for=state4>Go to state 4</label>
  </div>

  <div id=view2>
    <h2>View 2</h2>
    <label for=state3>Go to state 3</label>
    <label for=state4>Go to state 4</label>
  </div>
</form>
The following images illustrate the four different states of this example.

**View 1**

Go to state 2   Go to state 4

**View 2**

Go to state 3   Go to state 4

**View 3**

Go to state 1
Sending GET network requests upon a click may be useful. Browsers may intelligently load resources once they are needed. One example of this loading is with CSS-defined background image. Browsers may only load the image once the container the image is defined for is actually displayed. However, the browser may not know what is and is not an image, and may silently fail if the request fails or whatever is or is not returned is not an image. Thus, it may be possible to associate a network request with a particular state by setting the desired request as a background image URL.

<style>
input,
#view1,
#view2 {
  display: none;
}

div {
  padding: 5px;
  border: 1px solid black;
}
label {
  display: inline-block;
  padding: 5px;
  margin: 5px;
  color: white;
  background-color: black;
}

#state1:checked ~ #view1,
#state2:checked ~ #view2 {
  display: block;
}

#network_request {
  display: inline;
  opacity: 0;
  height: 1px;
  width: 1px;
  background-image: url('https://my.network.request?some=information');
}

</style>

<form>
The following images illustrate the two different states of this example.

**View 1**

Go to state 2

**View 2**

Go to state 1

In some instances, the browser may try the network request once.
There may be no restriction to having multiple state machines, or a state machine within a single UI view. Defining multiple forms and UIs inside those forms may be acceptable, and these forms can be used to define nested and arbitrary complexity inside the UI.

The techniques described above accomplish several goals such as the ability to define a finite state machine of arbitrary complexity and the ability to build a user interface that accepts clicks as inputs and can interface with the finite state machine to use these clicks to transition throughout the system without JavaScript.

The techniques described above may not be limited to uses where JavaScript is not available. In some cases, where a developer is looking to reduce the JavaScript payload size, these techniques may be a good option. In cases where better performance is desired, a developer may use these techniques to outperform JavaScript since these techniques can be accomplished entirely with CSS and native browser tooling.
ABSTRACT

To create a web page with state, a developer may not be required to use JavaScript. The developer may be able to use HTML and CSS to create a lightweight web page that can remember whether a user clicked on a web page or the location of the user click or selection without JavaScript. The developer may use checkboxes or radio buttons or both to create web pages with state without using JavaScript.