Supporting Empirical Modelling in the Browser

Speaker: Tim Monks

Mathematics BSc (University of Warwick)
Computer Science & Applications MSc (University of Warwick)

Software Engineer at Trigger
https://trigger.io/

What I do these days

- Framework and tools for making mobile apps using HTML/CSS and JavaScript
- Service for pushing out updates to users without requiring app store review
- Check us out at: https://trigger.io/

Overview of this talk

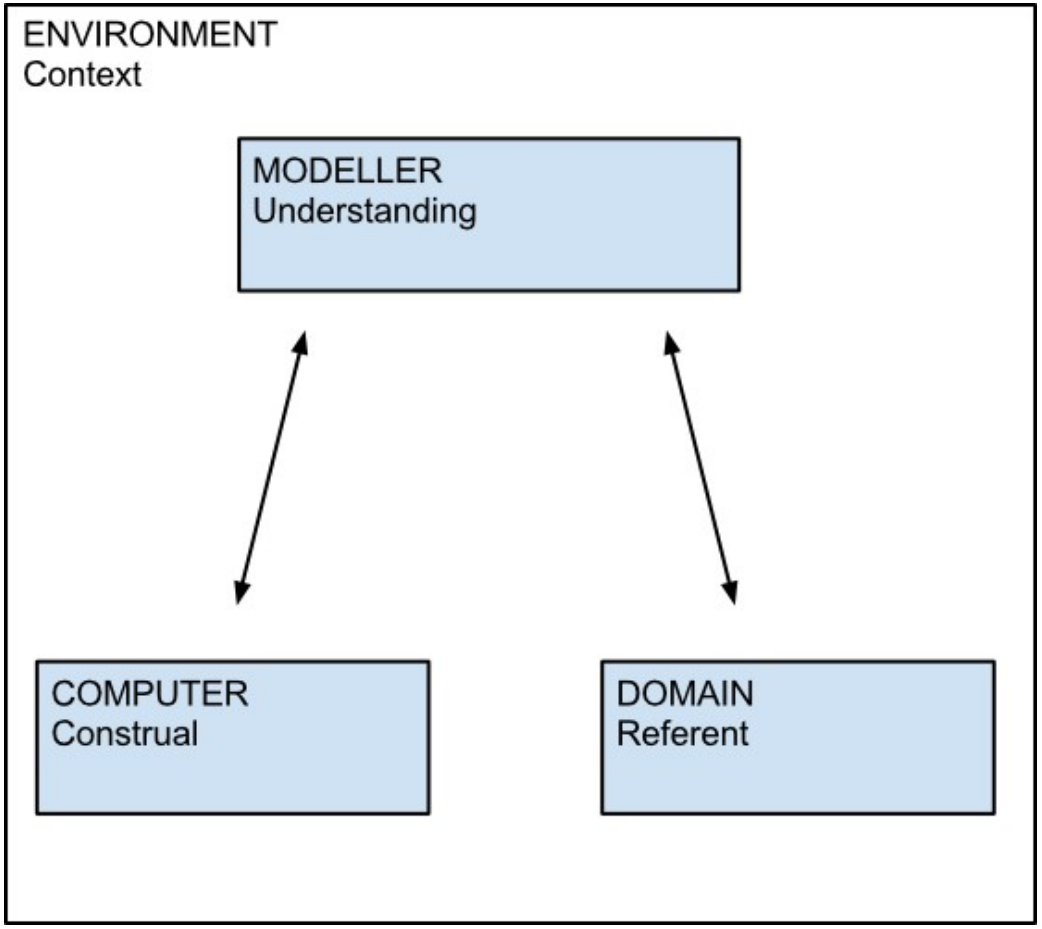
- The Why
  - My perspective on Empirical Modelling
  - Why bother with the browser?
- The What
  - Qualitative view of JsEden
  - Comparison with already existing tools
- The How
  - Available technological options
  - Overview of JavaScript
  - Architecture of JsEden
  - Implementing dependency
  - Implementing support for the EDEN like language
- Plans for further development
  - Issues that need to be resolved
  - Collaborative modelling
  - Persistance/versioning of models
  - JsEden in contexts other than the browser?

The Why

Where I try to convince you that this was worth doing at all

What do I think Empirical Modelling is about?

Isn't it obvious?



What do I think Empirical Modelling is about?

- Simples: Interactively building one thing to help reason about another thing

- We call these the **model** and the **referent** (the thing that the model refers to)

- In particular, how can we use computers for this task?

How do people do Empirical Modelling?

- Several tools that already exist can support the activity of Empirical Modelling
  - Spreadsheets
  - Tkeden (Modelling with definitive scripts)
    - EDEN (the language)
    - EDEN (the virtual machine)
  - Cadence (Modelling with process like definitions)
    - DASM (the language)
    - DOSTE (the virtual machine)
- The thing in common with all of these is the immediate effect of interactions with the model

Why I was interested in EM

Overall, I'm interested in anything that makes creation of interactive media on a computer more accessible to humans

- Constructivism
  - Knowledge is not a universal substance, but a personal construction gained through experience
- Computers
  - Had been learning to program for the past 3 years so was interested in creating software already
- Interactivity
  - EM seems to be about making the behaviour of software more tangible and relatable to everyday experience

Why try to do this in the browser?

- In my opinion it's easier for people to engage with a web page (less steps to getting there)
- Depending on choice of implementation there are other benefits from a development point of view and for the end user

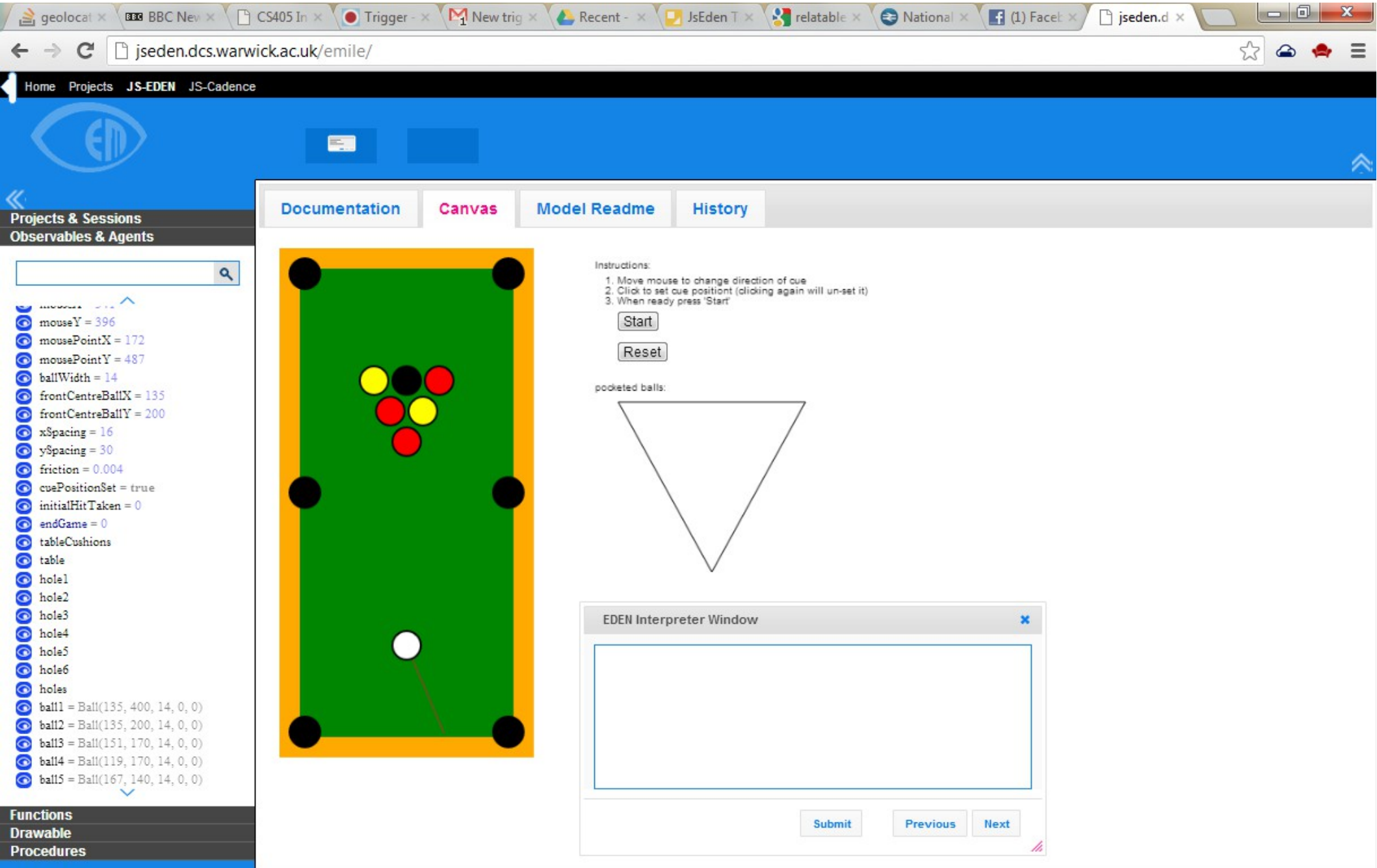
The What

Where I point out what it is that was actually done

Summary of JsEden

- JsEden is a tkeden like environment that you can use to create models in the browser
- The “JS” refers to the fact that the environment is built on top of JavaScript
- The “EDEN” part refers to the fact that models use a similar world view to EDEN, and supports an EDEN like language (unlike tkeden/dtkeden/webeden which all make use of the EDEN virtual machine directly)

User view of JsEden



How does this support EM?

- Permits the same method of model construction as tkeden: **modelling with definitive scripts**

- A model built through MWDS revolves around **Observation**, **Agency** and **Dependency**.

What is MWDS?

- Creation of a computer based model where:
  - All state is described as **observable** quantities
  - All change is due to **agency** (action) or **dependency**
  - A change in some observable due to dependency is **indivisible** from change in the observable it is defined in terms of

Examples of observable quantities

- The temperature of a cup of coffee
- The speed of a moving object

Examples of agency

- A speaker causing your boredom levels to change during a presentation
- A magnetic field causing movement of a paperclip across a desk

Examples of dependency

- The gravitational potential energy of an object is dependent on it's mass

How does JsEden compare to already existing tools?

- tkeden
  - Same conceptual framework, different implementation
- webeden
  - Same conceptual framework, both in browser, different architecture and implementation
- Cadence
  - Different conceptual framework (modelling through moment-to-moment transition functions)
  - Attempting to step into more visual mechanisms for model construction
- Spreadsheet modelling
  - Scripting language rather than grid, ability to define machine based agents (though VB may also permit this?)

The How

Where I explain how things were done, and how they might have been done

Main technical questions

Need to decide:

- Is the bulk of the dependency maintenance logic going to execute on the user's machine or as a service?
- What technologies allow us to execute code in the browser?

Where does the dependency maintenance code run?

Two possible approaches:

- Bulk of computation happens on a remote machine, changes to the model state are communicated to a thin client in the browser over a network.

- All computation happens on the client's machine. The user downloads the dependency maintenance engine which is somehow embedded into a web page.

Serverside

+ Can reuse already existing dependency maintenance software

+ Scalable both vertically and horizontally

- Requires internet connection throughout modelling

- Lag when waiting for updates to come over wire

- Increased infrastructure costs

Clientside

+ No network induced latency

+ Ability to work offline

+ Infrastructure is just hosting, so scalable and cheap

+ One user doesn't effect QOS for another

- May not be able to re-use already existing software (Need a supported runtime in the browser)

What technology can we use on the clientside

- Embedded Flash Application
- Embedded Java Application
- Other 3rd party plugins (e.g. NaCL, Unity)
- JavaScript

What technology can we use on the clientside

- Really it comes down to JavaScript vs an embedded application using a 3rd party plugin.
- JavaScript requires no plugins, and is rising in popularity (buzzword is HTML5)
- JavaScript works on more devices – tablets/mobile have poor or no support for Java or Flash based sites

I ended up choosing JavaScript

+ No plugins required

+ Great visualisation APIs, including the DOM

+ Bright future compared to most plugins

- Modern browser required (though IE6-8 can use Google Chrome Frame as a workaround)

- Performance? JavaScript has come a long way with e.g. JIT compilation, but Java is probably the best here. DOM interaction still overpowers everything anyway so probably irrelevant.

Other sort of reasons

- Personal preference/interests
- I could kind of see how it would work
- Other new languages built on top of JavaScript so I can benefit from other people's work

Wait, what's JavaScript?

- **It has nothing to do with Java** (except maybe some minor syntactic resemblance)
- JavaScript is code that can be included directly in a HTML document, to add logic to the page
- Capable of responding to events in the page (e.g. a mouse click)
- Capable of modifying the content of a HTML document through the DOM (Document Object Model)

Example JavaScript

```
<!doctype html>
<html>

<head>
<script type=”text/javascript”>
  window.onload = function () {
    document.body.innerHTML = 'Hello world';
  };
</script>
</head>

<body>
</body>

</html>
```

Example JavaScript

```
<!doctype html>
<html>

<head>
<script type=”text/javascript”>
  window.onload = function () {
    document.body.innerHTML = 'Hello world';
  };
</script>
</head>

<body>
</body>

</html>
```

Example JavaScript

```
<!doctype html>
<html>

<head>
<script type=”text/javascript”>
  window.onload = function () {
    document.body.innerHTML = 'Hello world';
  };
</script>
</head>

<body>
</body>

</html>
```

Example JavaScript

```
<!doctype html>
<html>

<head>
<script type=”text/javascript”>
  window.onload = function () {
    document.body.innerHTML = 'Hello world';
  };
</script>
</head>

<body>
</body>

</html>
```

Whirlwhind overview of the language

- Dynamic, weak, duck typing
- The important data types are:
  - Number, String, Boolean
  - Function
  - Object (and Array, which is really just Object)
- Garbage collected
- Allows reasonable mix of imperative, object oriented and functional styles

What do I mean by imperative style

- JavaScript programs are generally a sequence of statements:

```
var x = “Hello”;
x = x + “, World!”;
console.log(x);
```

- Mutable variables
- Explicit control flow

What do I mean by object oriented style

- Object type is basically a value with a user defined interface:

```
var o = {
  quack: function () {
    console.log(“quack”);
  }
};

o.quack(); // prints quack
o.bark(); // type error
```

- No compile time guarantees. Prototypal inheritance rather than class based.

What do I mean by functional style

- Support for first class functions:

```
var square = function (x) {
  return x;
};
console.log([1,2,3].map(square));
```

- Higher order functions hence possible and natural to define
- Not strictly mathematical functions though, no constraints on side effects

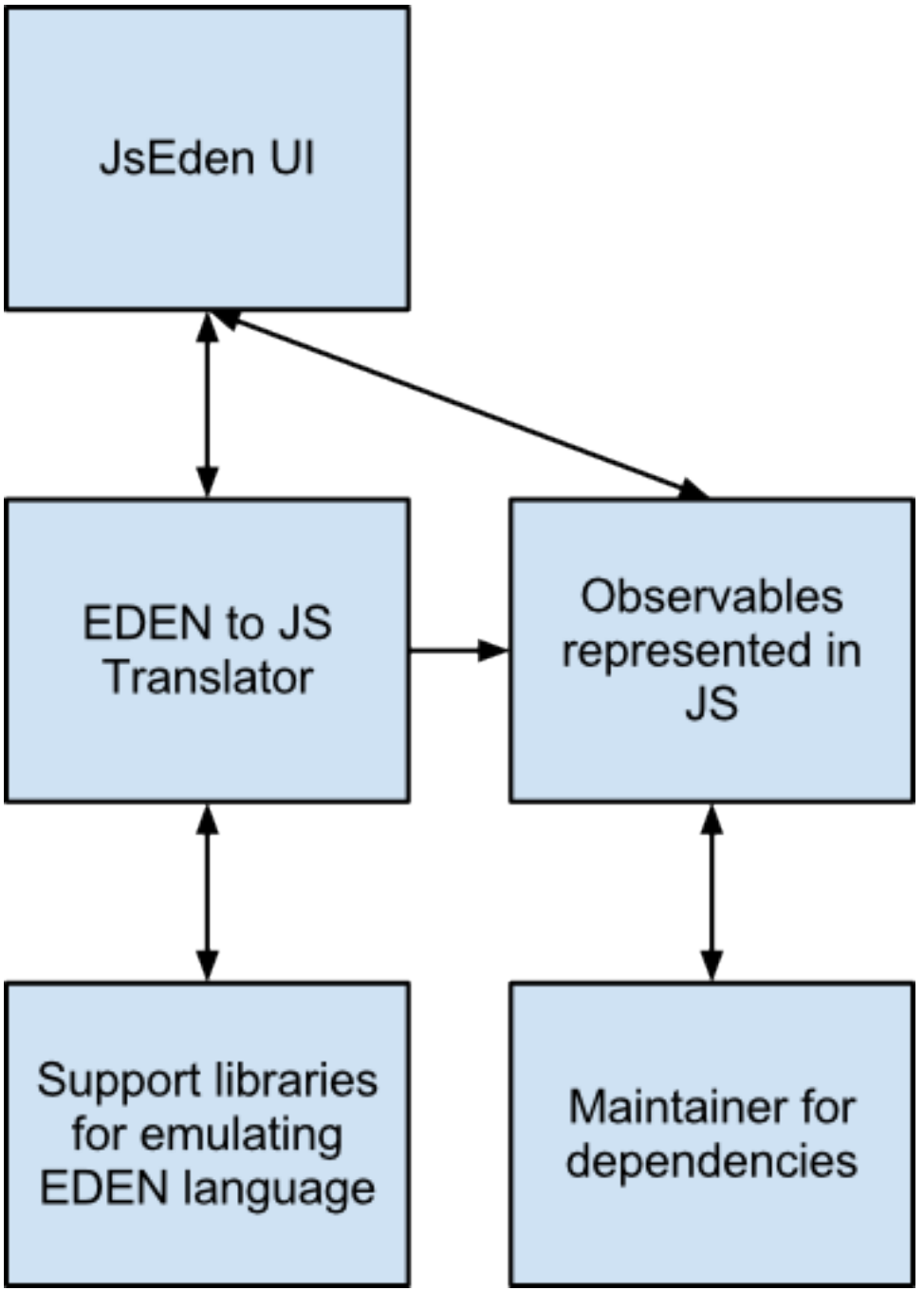
tl;dr

JavaScript is a surprisingly flexible language that allows you to add more interesting behaviour to web pages

Overview of JsEden architecture

- JavaScript library to bring support for dependency and EDEN style agents (aka. triggered actions)
- Translator to compile EDEN-like notation to JavaScript.
- Libraries to support creation of visual interfaces for user interaction.

Overview of JsEden architecture



Supporting MWDS in JavaScript

- Before we can translate EDEN to JavaScript we need to tackle the following:
  - How do we store the values of observables?
  - How do we store the definitions of observables described using a formula
  - How do we store EDEN style agents
  - How do we allow agents to observe model observables for change

Recap: What is dependency?

- Dependency is a way of reasoning about observations in change
- To an observer, changes in certain observables causes **indivisible change** in others
- While modelling, this comes down to defining one observable as a formula referring to other observables
  - A is B + C;
- Need to have a clear idea about what the observers are and what the observables are before we can implement dependency

What can we consider to be an observer in a JsEden model

1. Human observers of a model
   - Observation of external behaviour

2. EDEN style triggered actions
   - Observation of observables for changes

What can we consider to be an observable in a JsEden model

- We could try to store the values of observables in regular JavaScript mutable variables
- Problems:
  - Need notification about changes
  - Need support for atomic transitions of more than one observable

Symbol table data type

- Rather than trying to use the JavaScript symbol table as our set of observables, use a value to represent the set of observables for a model
- Basically just a hash table where observable names are the keys, mapping to a structure containing the value and metadata for an observable

Observable data type

- Object type that stores the value for the observable
- Stores both the symbols that this observable depends on and vice versa
- If defined using a formula, stores this formula as a function, which is evaluated when dependencies change

More detail on these types in JsEden

- The symbol table stores observables in a hash table, and exposes a method for getting access to them:

```
var symbols = new SymbolTable();
var temperature = symbols.lookup('temperature');
```

More detail on these types in JsEden

- The observable type exposes methods for querying and updating the value of an observable:

```
var temperature = symbols.lookup('temperature');
temperature.value(); // currently undefined
temperature.assign(21); // update the observable
```

More detail on these types in JsEden

- The observable type exposes methods for definition:

```
var potential = symbols.lookup('potential');
var mass = symbols.lookup('mass');
potential.define(function () {
   return mass.value();
});
potential.dependsOn(mass);
```

More detail on these types in JsEden

- The observable type exposes methods for observation:

```
var watcher = symbols.lookup('watcher');
var mass = symbols.lookup('mass');
watcher.assign(function () {
   console.log(“Mass changed to: “, mass.value());
});
watcher.observes(mass);
```

How is the EDEN language supported?

- Simply translate from one EDEN language to JavaScript
- Translator is just a JavaScript function so easy to include in the JsEden environment
- Translator is defined using a BNF and a corresponding translation function is generated using a parser generator (called Jison)

Fin