JS-EDEN-SVG: Extending the range and capability of JS-EDEN observables

Bernard Sexton

January 29, 2013

Abstract

This project considers an SVG implementation of the JS-EDEN [1] environment in pursuit of enhancing the notion of communicable representation or construal conveyed by models defined in the browser environment. The motivation for doing this is grounded in improving how models are construed by the user through improvements of and modifications to artefact representation in JS-EDEN. The identification of differences between the current HTML5 implementation and an SVG variant is then highlighted by modelling an existing JS-EDEN project to identify how artefact representation affects the overall construal of a model and how the future development of JS-EDEN benefits from the use of standardised and widely available tools.

1 Introduction

JS-EDEN is a definitive environment for the browser modelled on the concepts of observables, dependency and agency of Empirical Modelling, supported by the EDEN (Engine for Definitive Notations) environment [1]. The tool combines the use of JavaScript and HTML5 to allow for the development of environments and their models in a way which supports the notion of a construal or 'communicable representation' similar to tools such as tkEden and more recently Web Eden [2]. Web Eden provides the basis for Monks justification of a JavaScript based 'definitive environment.' This is because Web Eden's reliance on an remote instance of the tkEden dependency engine meant that the observable capacity of construals were limited by the inefficiencies and perceived 'lag' of the Web Eden architecture. A significant benefit of JS-EDEN is its compatibility with a number of browsers and contexts as well as it's use of a JavaScript based interpreter and maintenance engine for the preservation of ODA for new and existing models. However, the latest *emile* variant supports a limited subset of objects meaning the expressiveness of models are reduced by the lack of drawables and parameters that can be used to define them in the browser environment. This limitation of the current HTML5 implementation in JS-EDEN to fully express the characteristics of a model's constituent parts, for example the particular transformations or groupings of objects with similar semantic meaning or association that make up an artefact can influence the way in which the original referent is construed by the user. The interactive graphics represented using the definitive programming paradigm in JS-EDEN aim to express the geometric relationships between definitions in a way which closely represents the referent as the author intended. The technical issues present in the evaluation of a definitive platform for interactive graphics are still relevant today [3]. SVG provides a framework which contributes to the discussion on the benefits of definitive notations being used to represent the dependencies between data.

2 Scope

I aim to replace the existing HTML5 Canvas that is used to represent visual artefacts with an SVG based alternative. The existing entities used to represent shapes and HTML based components will be replaced with explicit, XML based, SVG definitions that provide basis for the construction of models and their artefacts. Existing functions for transfor-

mations will be reimplemented and extended with additional transformations. The ability to group definitions as well as the ability to re-define the global co-ordinate system of the SVG viewport and instantiate local co-ordinate systems will also be maintained. Additional means for establishing more fluid agency in terms of animation are considered along side existing agency in terms of the capabilities of the Eden notation. This redefinition of the JS-EDEN tool is carried out with a view to evaluate how the capability and accuracy of the supporting graphics framework influences the ability of the definitive notation to support accurate construal of the author's intended referent through the concepts of ODA. This work indirectly supports one of the further work points that Monks specifies in his dissertation, namely point 9.5 'Modelling with Hierarchical Structures' [1]. SVG supports a document tree of elements meaning that observables would be ordered in a tree hierarchy depending on their dependency with other SVG elements. Theoretically watches can be applied to observables to check whether they are present in the tree or not after a given operation as is alluded to in Monks thesis.

3 EM and SVG

JS-EDEN provides a suitable platform to develop an SVG based drawing library for modelling activity but SVG isn't just constrained to the browser with it's use well documented in software for the design of engineering drawings and is implemented in some commercial CAD packages for the development of architectural drawings [5]. SVG is a vector based image format that contains support for interactivity and animation, it's XML markup and meta data support means that SVG drawings can be searched and indexed directly. SVG also supports direct and indirect scripting using JavaScript [7].

Modelling activity using EM tools allows for the development of artefacts where the personal experience of situations or particular subject matters is contained. The technical execution of user interactions between Canvas and SVG differs significantly . The event model and user interaction supported by Canvas is coarse where all activity that the user has with the canvas are based on manually programmed mouse coordinates. Indeed, in various models in the JS-EDEN library, significant computation is dedicated to the problem of identifying where the interactive regions for particular artefacts begin and end [9][10]. This can become hard to track and is open to unexpected behaviour especially if aspects of the canvas environment change. The event model and user interaction for SVG is at the object level with each artefact drawn within an SVG container representing a primitive graphic element such as a line, rectangle or path [4]. This means that interaction can be programmed and assigned to a specific element reference rather than just a region in the canvas. This means that we can implement different types of interaction based on elements in a identified grouping or on different mouse keyboard events without the need for complex region identification.

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="100" cy="50" r="40" stroke="black"
  stroke-width="2" fill="red" />
</svg>
```

Figure 1: SVG markup for the rendering of a circle

SVG better supports a range of primitive and complex shape types which can deliver more accurate representation of artefacts as definitions are revised and refined allowing the user to gain a better understanding of a model. Elements with similar semantic meaning in the context of a given model for example could be grouped together using the `<g>` tag. Their existing definitions would be preserved but they would also have a definition in relation to other elements in their grouping. SVG also contains features for the painting and filtering of elements. Primitive types can be subject to the styling attributes of the CSS specification but also attributes specific to SVG; the `marker` element for defining the directedness of a graph with the use of arrowheads, `clipping` and `masking` for obscuring parts of an element or a group of elements or `gradients` and `filters` for defining depth of field or emphasis which can be applied to elements, groups and viewports. The number of features that SVG possesses is important to sound interpretation

of the construal but it's well defined, static definition with attributes that can change dynamically through agency via an EM tool or otherwise aligns the drawing framework itself more closely than the HTML5 canvas to the concepts of observables, dependency and agency. The SVG elements themselves are the observables we can refine and revise and within or outside of the context of an EM tool where the SVG defined model can become a standalone iteration of the interaction a user has had with a model at a particular point in time. SVG can be subject to state change much in the same way as the HTML canvas but it can also preserve the state of visualised observables in a manner independent of a given EM tool. JS-EDEN is a suitable platform for an experimental implementation of this alternative visualisation paradigm for models.

4 Applications to JS-EDEN

Previous tools have used the separate Donald line drawing notation for the development of artefacts to support modelling activity [8] but the JS-EDEN environment utilises the fact that the Donald notation is merely a definitive representation of an underlying EDEN function which provides interactive graphics in tools such as tkEden for the drawing of lines. Monks uses the notion of prototype functions in JavaScript and the Eden `func` construct to allow access to a set of primitive shape and mathematical functions. A `picture` function is also defined in order to display specified definitions.

```
openshape a
within a {
line l1, l2...
```

(a)

```
s is Shape(a,b,c,d)
picture is [s]
```

(b)

Figure 2: Donald syntax and equivalent JSEDEN syntax for a primitive

We define the observables in the usual definitive manner, `variable = value` which assigns the function name and parameter values to the defined variable name. These observables are then updated using the drawing library based on any dependencies they may have or agency that may be executing upon them. The definition and redefinition of components on the canvas is executed on the `context` property of the HTML Canvas meaning that the 'observables' that are to be defined in the picture are redrawn on each call of the `picture` function. An SVG based implementation defines drawing elements within the document tree of the webpage itself but does so dynamically and at a level of the tree which does not affect the rest of the content in the DOM. This would mean, much like how we can refine observables and observe changes, our modification of SVG based artefacts would result in *actual* real-time changes of the attributes of the observable rather than being refreshed and redrawn. It also means that we can utilise the event based programming model of JavaScript to directly operate on visualised observables within the DOM using procedures for specific mouse events on the elements themselves rather than just the regions surrounding them.

5 Development

The development of the SVG variant of JS-EDEN was premised on the *emile* variant. This relatively stable version of the tool contains an existing set of functions and procedures for drawing on the HTML5 canvas but demonstrated some unexpected results when experimenting with some of the existing models.The `canvas` observable would often remove existing operations when the surrounding window was resized and the values for this resizing would sometimes not permeate though to the relevant observable reference in the symbol table. Due to the lack of documentation and sometimes obtuse error message reporting from the JavaScript maintainer, the new `canvas` observable was redefined in terms of the width and height specified by the user and not by the dimensions defined by proximity of the canvas element to other divs.

The `svgcanvas` replaces the existing HTML5 canvas. Unlike the previous canvas object which operated on the `context` object, the SVG canvas is subject to a number of operations in order to display or maintain the definitions of new or existing elements respectively. These operations are used in the prototype function of each shape primitive or filter definition.

- *createElementNS* creates the element definition

- *getElementById* gets existing element
- *setAttribute* sets attributes of element
- *appendChild* appends element to the svg canvas

Existing Definitions

All existing definitions have been re-implemented in SVG and now have style tags to address the inconsistencies between some elements having styling capability versus limited color functionality. The *polygon* definition uses SVG grouping to group the points of the polygon together so that operations such as transform etc can be applied to the element globally. The *text* definition now has font capabilities with SVG being able to inherintely support non-unicode character types regardless of the language support of the host machine. The Arc function has also been redefined and a Bezier function introduced to leverage the powerful path rendering capabilities of the SVG path construct.

New Definitions

Inspired by the Dependency modelling tool in tkEden by Harfield et al [11], the *Path* definition has been implemented with markers to indicate the directedness of the relationships between nodes. The path is defined by a start and end node referenced by observable name. Each path can be assigned a label indicative of the weight of that particular path or other metric. Triggers can the be applied to elements directly for a number of different events allowing for a representation of the DMT to be possible in JS-EDEN.

The concept of grouping elements according to a semantic meaning or location in the model is also introduced through the use of the layer function. The layer function makes use of the `<g>` construct in SVG to group specified observables by their name. These groupings can be revised in a similar fashion to the refining of conventional shape primitives. This allows for functions such as transformations or particular triggered actions to be applied to a set of observables. Grouping of elements or having multiple group containers allows for the concept of co-ordinate systems. The groups can be

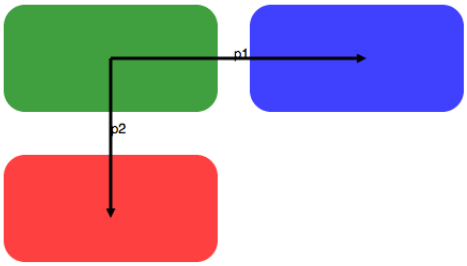

Figure 3: Directed dependency visualised using the new `path` function

defined in terms of their x and y location as well as their width and height. The layers can then be translated around the canvas with the elements contained within them referencing their (x,y) positions in terms of the group. This allows for models which make use of mapping data or graphing as they can use co-ordinates for observable rendering in isolation of the global size and specification of the SVG canvas. New functions for translating observables have also been implemented with skew being added, translations can also be applied to grouped observables. SVG Filter functions have been defined to add additional components such as light sources and shadows to elements. These functions can also be redefined to produce different effects.

6 Testing

Testing of the new SVG variant of JS-EDEN was incremental. As each function was re-implemented or a new function added, it was first tested upon it's initial definition and then tested again upon it's redefinition to ensure that the SVG elements were being actively maintained by the observable references in the symbol table. The SVG elements were also tested in the context of loading in complete models with revised definitions for some shape primitives. These brought up some issues which will be discussed later.

6.1 Revisiting Jugs

To test the capabilities of the reimplementation, I adapted the Jugs model by Beynon to use the new SVG based primitives as well as additional filters

Figure 4: The Jugs model, re-implemented in SVG.

and transforms supported by the SVG framework. The results demonstrated that the relevant observables and dependencies were maintained in lieu of the new SVG drawables and that additional agency could be implemented as a result of the increased functional scope of the SVG library. The frame rate recorded for the rendering and agency aspects of the model as slightly lower (statistic) than the HTML5 equivalent but when scaled up, the greater number of pixel manipulations in the HTML5 canvas compared to the vector scaling operation meant that the SVG representation achieved a superior frame rate.

6.2 Master variant of JSEden

During the development of the SVG variant of JS-EDEN, Nick Pope revised the emile version to what is now termed as the 'Master' version of JS-EDEN. Nick's effort took the existing interface replacing it's components with 'plugins' and 'views' where each drawing function is forms part of the HTML5 canvas plugin. This can then be loaded through the use of a view. This pattern of usage provides a basis for 'hot-plugging' new plugins such as an SVG canvas viewer into JS-EDEN while maintaining the existing Eden engine. Due to time constraints I was unable to refactor the emile work into this master variant but this exercise would be fairly simple given the ability to create new functions as part of the drawing plugin and then dynamically calling the SVG plugin along with an appropriate view.

6.3 Outstanding issues

There are still some issues with using an SVG in the current emile version of JS-EDEN that are mainly related to the physical representation of observables. In HTML5 Canvas, our observables which relate to shape primitives are not physically represented in the canvas and there is no reference back to the observable until the observable is modified and the canvas redrawn. Under an SVG paradigm, each observable that contains the value of a shape primitive is physically defined in the canvas and is referenced by the observable identifier. However, due to the way in which the picture parameter is defined, there is no observable identifier passed when the picture function is updated, this means that identification is only currently available through the interpreter which uses a 3rd party plugin for the text input window. This 3rd party plugin has been shown to sometimes truncate input and hence not execute functions when desired meaning it serves as an unreliable means for identifying the appropriate observable references.

Some existing models in JS-EDEN make extensive use of jQuery in the manipulation of observables, dependency and agency. Sometimes functions and procedures are defined in terms of the EDEN `proc` and `func` syntax but then utilise JQuery and JavaScript to access observable values and redefine them. This can sometimes result in issues with the definition of observables with drawing definitions where exceptions are thrown or particular artefacts of the model fail to render.

7 Future Work

SVG provides a drawing framework with a range of primitives and effects that can be used to represent artefacts of models in a way which allows a user to develop an arguably more meaningful understanding of a given construal given it's greater depth of representation and interactivity. The next step for JS-EDEN is to further support more complex models with many observables, dependency and complex agency such as those observed in the malaria model [**?**] with efficiency while retaining the ability to experiment with each aspect of the models in the environment to maintain exploration and personal understanding of the subject matter. This could be achieved by using a hybrid visualisation framework that uses HTML5 for intensive pixel based computation and SVG for the representation of specific elements where efficient scaling and frequent refinement of the observable value is required. Packages such as `Raphael.js` provide JavaScript capability

for producing SVG graphics without the need for direct interaction with the DOM which would make maintenance and implementation of future functionality easier given the extensive documentation of the library.

8 Conclusion

JS-EDEN is one of a number of tools that is contributing to the current direction of Empirical Modelling, providing a platform that uses a single definitive notation with which to develop models that exhibit the concepts of observables, dependency and agency. The SVG implementation of JS-EDEN has led me to evaluate the appropriateness of an XML based drawing framework in the context of a system which requires frequent maintenance of the definitions of the XML attributes as well as their dependencies between parent elements and other elements in a document tree representative of a given model. While SVG provides an alternative means for representing these models, it has been observed the continuous population and updating of the document tree leads to noticeable increases in load time when compared to models populated in their HTML5 canvas equivalent. With the introduction of the more stable master version of JS-EDEN, it may be more appropriate to introduce a drawing framework that is capable of using both SVG and HTML5 Canvas [12] to produce models capable of complex visualisation and scalability.

9 Acknowledgements

I would like to acknowledge Meurig Beynon and Nick Pope for their continued guidance, support and development in relation to the implementation of an SVG variant of JS-EDEN through discussions on the js-eden Google group and in person.

10 Weighting

Paper: 60%, Implementation: 40%

References

[1] Monks, T. (2011). *A definitive environment for the browser.* Unpublished masters thesis.

[2] Beynon, M; Myers, R; Harfield, A. (2009) *Web Eden: support for computing as construction?* In: International Conference on Computing Education Re- search (Koli Calling), Koli National Park, Finland.

[3] Beynon, M. (1989). *Evaluating Definitive Principles for Interaction in Graphics* in New Advances in Computer Graphics. Proceedings of CG International '89.

[4] W3 Commission. (2011). *Scalable Vector Graphics (SVG) 1.1 (Second Edition).* (url: http://www.w3.org/TR/SVG/). (accessed on 27/01/2013).

[5] Fahiem,M; Farhan, S. (2007). *Representation of Engineering Drawings in SVG and DXF for Information Interchange* 6th WSEAS International Conference, Cairo, Egypt.

[6] Rungrattanaubol, J. (2002). *A treatise on Modelling with definitive scripts.* PhD thesis, University of Warwick.

[7] Kalb, M. (2012). *SVG-based Knowledge Visualization.* Unpublished Diploma Thesis.

[8] Beynon, M. (1986). *DoNald: a definitive notation for line drawing.*

[9] Nixon, J (2011). 2-Ball & 7-Ball Eightball JS-EDEN Model.

[10] Cranman, M. (2011). Parallel Parking - JS-EDEN Model.

[11] Wong, A. (2003). *Before and Beyond Systems: An Empirical Modelling Approach.* PhD thesis, Department of Computer Science, University of Warwick.

[12] Sucan, M. (2011). *SVG or Canvas. Choosing between the two.*[Online](url: http://dev.opera. com/articles/view/svg-or-canvas-choosing-between-the-two). (accessed on 28/01/2013)

A SVG Jugs Representation

```
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
version="1.1" id="svgcanvas" width="100%"
height="100%" currentscale="1">

<foreignObject id="but1" x="50" y="230" width="100"
   height="50" style="left: 50px; top: 230px;">
        <input type="button" id="but1" value="1:Fill A
           ">
</foreignObject>

<foreignObject id="but2" x="120" y="230" width="100"
   height="50" style="left: 120px; top: 230px;">
        <input type="button" id="but2" value="2:Fill B
           " disabled="disabled">
</foreignObject>

<foreignObject id="but3" x="200" y="230" width="100"
   height="50" style="left: 200px; top: 230px;">
        <input type="button" id="but3" value="3:Empty
           A" disabled="disabled">
</foreignObject>

<foreignObject id="but4" x="300" y="230" width="100"
   height="50" style="left: 300px; top: 230px;">
        <input type="button" id="but4" value="4:Empty
           B">
</foreignObject>

<foreignObject id="but5" x="390" y="230" width="100"
   height="50" style="left: 390px; top: 230px;">
        <input type="button" id="but5" value="5:Pour">
</foreignObject>

<g id="statuslabelg" font-family="Verdana" font-size
   ="15">
```

```
        <text id="statuslabel" x="250" y="150" fill="
            black">Target is 4 :  awaiting input</text>
</g>

<line id="jugA_left" x1="50" y1="200" x2="50" y2="140"
     stroke="black" stroke-width="2"></line>

<line id="jugA_right" x1="110" y1="200" x2="110" y2
    ="140" stroke="black" stroke-width="2"></line>

<line id="jugA_base" x1="50" y1="200" x2="110" y2
    ="200" stroke="black" stroke-width="2"></line>

<line id="jugB_left" x1="140" y1="200" x2="140" y2
    ="100" stroke="black" stroke-width="2"></line>

<line id="jugB_right" x1="200" y1="200" x2="200" y2
    ="100" stroke="black" stroke-width="2"></line>

<line id="jugB_base" x1="140" y1="200" x2="200" y2
    ="200" stroke="black" stroke-width="2"></line>

<rect id="jugA_water" x="50" y="200" rx="0" ry="0"
    width="60" height="0" style="fill:blue;opacity
    :0.40"></rect>

<rect id="jugB_water" x="140" y="100" rx="0" ry="0"
    width="60" height="100" style="fill:blue;opacity
    :0.40"></rect></svg>
```