A Dynamic Hierarchical Approach to Modelling and Orchestrating the Web of Things Using the DOM, CSS and JavaScript

Alex Owen
Web Science
University of Southampton, Southampton, UK
alex.owen@soton.ac.uk

Kirk Martinez
Electronics and Computer Science
University of Southampton, Southampton, UK
km@ecs.soton.ac.uk

ABSTRACT

There is a lot of work in progress by the W3C and others surrounding a Web standards compliant Web of Things (WoT) which it is hoped will unify the current Internet of Things infrastructure. Our contribution to this uses the Document Object Model (DOM) to represent complex physical environments, with a CSS-like syntax for storing and controlling the state of 'things' within it. We describe how JavaScript can be used in conjunction with these to create an approach which is familiar to Web developers and may help them to transition more smoothly into WoT development. We share our implementation and explore some of the many potential avenues for future research. These include rich WoT development tools and the possibility of content production for physical environments.

CHI'19 Extended Abstracts, May 4-9, 2019, Glasgow, Scotland, UK.

ACM ISBN 978-1-4503-5971-9/19/05.
DOI: https://doi.org/10.1145/3290607.3312990

KEYWORDS

Web of Things; WoT; Internet of Things; IoT; Web Development; Document Object Model; Linked Data

```
<context id="living_room" class="room">
    <group id="table_group">
        <thing id="chair_1" class="chair" type="https://localhost/td/chair.jsonld" />
        <thing id="chair_2" class="chair" type="https://localhost/td/chair.jsonld" />
        <thing id="chair_3" class="chair" type="https://localhost/td/chair.jsonld" />
        <thing id="chair_4" class="chair" type="https://localhost/td/chair.jsonld" />
        <thing id="table" class="table" type="https://localhost/td/table.jsonld" />
    </group>
    <thing id="cooker">
        <thing id="hob" type="https://localhost/td/gas_hob.jsonld">
            <thing
                id="burner_1"
                class="burner heat_generating gas_powered"
                type="https://localhost/td/gas_ring_small.jsonld" />
            <thing
                id="burner_2"
                class="burner heat_generating gas_powered"
                type="https://localhost/td/gas_ring_small.jsonld" />
            <thing
                id="burner_3"
                class="burner heat_generating gas_powered"
                type="https://localhost/td/gas_ring_small.jsonld" />
            <thing
                id="burner_4"
                class="burner heat_generating gas_powered"
                type="https://localhost/td/gas_ring_big.jsonld" />
        </thing>
        <thing id="oven" type="https://localhost/td/oven_domestic_electric.jsonld">
            <thing
                id="heating_element"
                class="heat_generating electricity_powered"
                type="https://localhost/td/heating_element_1800w.jsonld" />
            <thing
                class="light electricity_powered"
                type="https://localhost/td/bulb_led_5w.jsonld" />
        </thing>
    </thing>
    <thing id="fridge" type="https://localhost/td/fridge_home_large.jsonld">
        <thing
            class="light electricity_powered"
            type="https://localhost/td/bulb_led_5w.jsonld" />
        <thing
            class="compressor electricity_powered"
            type="https://localhost/td/fridge_compressor_200w.jsonld" />
    </thing>
    <thing
        id="ceiling_light"
        class="ceiling_light light electricity_powered"
        type="https://localhost/td/bulb_led_22w_rgbw.jsonld" />
</context>
```

Figure 1: A context showing things, groups of things, and Thing Descriptions linked to thing elements.

1 INTRODUCTION

The Internet of Things (IoT) currently lacks coherence, especially when orchestrating on a large scale and across several proprietary ecosystems. Kleinfeld et al. [4] give a good overview of the problems of diverging approaches and the need for a Web standards compliant solution.

The World Wide Web Consortium (W3C)'s Web of Things (WoT) group has produced a specification to overcome the fragmentation of the IoT market [8] which is based on existing Web standards. The core of this links a 'thing' to a Thing Description (TD) [3]. A thing is often a physical device, but may also be virtual, a service or composed of many parts. The TD is a JSON-LD document containing the properties a thing can have; actions it can perform and events it can produce. TDs can be stored on the device or anywhere else on the Web where they can be accessed by a URI.

This specification provides a basis for describing things within a physical environment. Currently the representation is not one that content producers and Web developers are familiar with, as it is only just at the point of Candidate Recommendation [9] and a long way from industry implementation. We believe a large number of future WoT developers will come from the pool of over 21 million developers working on Web based projects [1] due to the overlap in skills required. Easing this transition will be an important challenge to achieve the eventual acceptance of the WoT concept.

We are working to produce an alternative representation for things using the Document Object Model (DOM). This can be used in conjunction with the W3C's WoT specifications, CSS and JavaScript to allow Web developers to apply their existing skill set, tools and libraries to create content for the WoT.

2 RELATED RESEARCH

Miorandi et al. [5] discuss many challenges of the IoT, including the digital representation and identification of things. Our approach tackles representation by using the DOM and linked data to describe things and their affordances. They also discuss the potential for referring to classes of devices, which we mirror in our use of user and machine defined classes on DOM elements.

There has been previous discussion of Generalized Cascading Sheets (GCS) being developed as a parent to CSS [7], however the concept has yet to progress from the discussion phase. In the initial dialogue, Håkon Wium Lie, one of the original proposers of CSS, agreed "it would be helpful to have a generic language for augmenting structures in a compact manner, like CSS does". GCS would allow for a more open and flexible approach which is not limited to the closed set of CSS properties. This would enable the use of the syntax in different domains beyond document styling. While this has yet to materialise, we see a benefit in exploring CSS-like syntax in other areas.

One specialist domain usage of CSS syntax was put forward by Martin Schuhfuss at JSConf 2015. He suggested using CSS to replace DMX [6], although the idea was never developed beyond a proof of concept and a live demo. His example shows the power of a CSS-like syntax to replace DMX commands, although this is a limited set of properties in a tightly controlled environment.

```
<style>

    /* The default stylesheet for the room */

    /* By default, everything is off for safety */
    .electricity_powered {
        power: off;
    }

    .gas_powered {
        level: 0;
    }

    /* The fridge over-rides the 'power: off;' above */
    #fridge .compressor.electricity_powered {
        power: on;
    }

</style>


<style>

    /*
     * A second, user selected style sheet sets the room up
     * for cooking using #burner_4 and the oven
    */

    .ceiling_light {
        power: on;
        color: #ffd080;
        brightness: 100%;
    }

    #burner_4 {
        level: 50%;
    }

    #oven .heating_element {
        power: on;
        temperature: 180deg;
    }

    #oven .light {
        power: on;
    }

</style>
```

Figure 2: Two CSS(T) style sheets for the context given in Fig. 1.

Hylli et al. [2] discuss the use of CSS selectors to identify IoT devices in a more open environment and their implementation has some similarities to the W3C's TD model. However, their system is based on the OpenAPI specification, which while free to use, is not a Web standard.

3 METHODOLOGY

We have created a system which represents a physical environment or 'context' and the things within it as a hierarchy in a similar way to the DOM represents a Web document. This DOM is held either within a central server or it can be distributed across several servers and clients. The server may run in a datacentre or locally using a hub, as long as it is accessible by a URI or IP address.

The context may be physical, such as a room, or more abstract, like things belonging to a person. It can be static, like the furniture in a room, or more fluid like the contents of a backpack. An example of a complete context can be seen in Fig. 1.

A developer could manually add nodes to a context to represent things and organisational groupings, or they can be automatically added by the things themselves as they are discovered or register themselves with a server.

'Thing' nodes each represent a concrete object or device within a room, such as a speaker or a cooker. They can also represent a part of an object: the light in a fridge or a cushion on a sofa. They can be placed within other thing nodes to represent strong relationships such as containment or direct proximity, such as a carton of milk in a fridge or a book on a table. Thing nodes can be linked to the W3C's TDs or other sources to add richer information about their affordances and attributes; this allows developers to gain more information about the device they are controlling.

Organisational 'group' nodes do not directly represent anything physical in an environment, but allow developers to organise things contextually, for example four chairs and a table may form a group, and a pair of speakers and an amplifier another. This can be used to show that the things are loosely related but not parts of a parent thing.

With an effective DOM in place to represent the strong and loose relationships between things, other technologies become available to make a more complete development experience. Using the DOM, a thing node can be assigned an ID and classes in much the same way as nodes in an HTML document. These can be either added directly by the developer or automatically from a TD or other linked data source based on specified criteria. Things could then be referenced using CSS selectors in the same way as elements of a Web page, for example, *hub.querySelectorAll('.light')* would select all things with a class of 'light'.

The structure and syntax of CSS can be borrowed to produce a CSS for Things, or CSS(T), where available properties can be dynamically discovered from the thing's TD or other linked data source. CSS(T) represents the thing's state, as well as the state it shares with other things of the same class. Two examples can be seen in Fig. 2. The CSS(T) properties map well to the TD's properties, and these can be used to retrieve additional information about them from any linked data associated with the property.

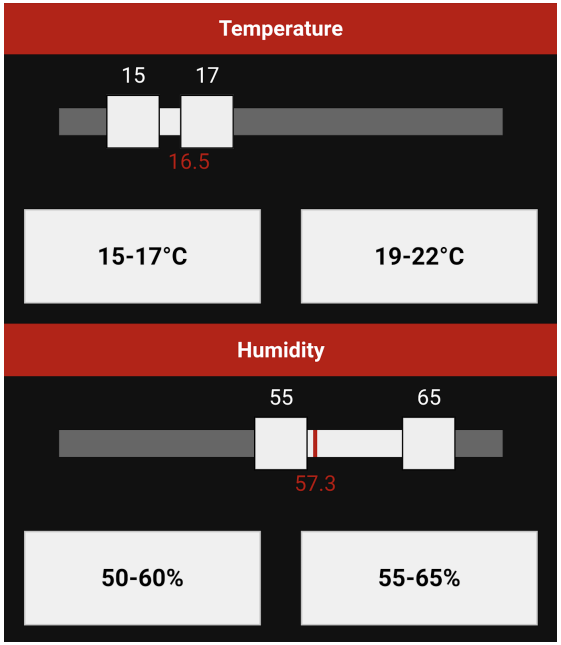

Figure 3: The simple Web UI controlling a heater, fan and dehumidifier, and showing data from temperature and humidity sensors.

Additionally, JavaScript can be used to control and manipulate the DOM and CSS(T) properties in the same way as it is used on the Web. Nodes can be selected using CSS selectors or directly by ID, then changed, modified, deleted or have other actions performed on them.

4 IMPLEMENTATION

Our system consists of a JavaScript client library and a server running on Node.js. The software runs on a Raspberry Pi in our development environment. The server stores the DOM and the active CSS(T) sheets. We have tested a few simple scenarios, and we plan to use it in larger ones soon.

These demonstrations include a light bulb and switch, a more complex lighting system with multiple coloured LEDs, and a climate control system for a small room with a Web UI, as seen in Fig. 3. Some of the physical devices we have built can be seen in Fig. 4.

This first implementation allowed us to test the principles but we do not anticipate significant technical issues with scaling it to larger environments.

One current limitation of our system is that each client represents a top level 'thing' within a 'context' and it must then register its components in the DOM. This means the developer cannot re-order the DOM manually; the DOM is automatically generated. Also, the components of a thing cannot register themselves in the DOM, i.e. currently a fridge can register the light inside it but the light cannot register itself as a child of the fridge. We plan to amend this in future revisions.

5 PRELIMINARY FINDINGS

We have found that technologically the idea is sound. Our system can record an environment as a DOM, apply IDs and classes to nodes, assign properties using CSS(T) and control the things it represents using JavaScript in a way that is similar to the code in everyday use by Web developers.

We have yet to test it within large, complex environments, however the underlying technology is the same as that used for the Web, which can handle many thousands of nodes.

One of the key issues with using a DOM built by dynamic discovery is that it will not include devices if they are not accessible by the server. Discovery may fail or the devices may be offline or not yet installed. TDs can go some way to solving this problem as they can be stored online on a highly available server, or copied locally to allow developers to stub or simulate things.

CSS(T) presents another problem because it does not have a limited list of properties as CSS does. Instead they are acquired from devices in the environment or from a simulation. This may create usability issues for the developer where manufacturers use different property names to describe the same attribute, or conversely, the same property name to describe different attributes. We believe this can be helped using semantic data linked to the property to provide context to the developer, and could take advantage of ontology matching to link equivalent or similar properties together.

We have talked informally with several developers and the feedback has been largely positive, and more formal, larger scale tests are currently planned.

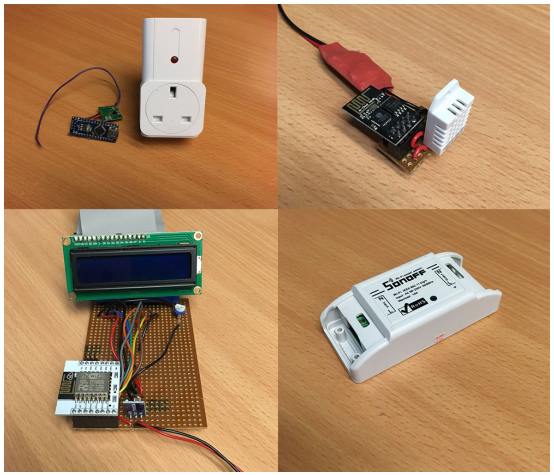

Figure 4: Clockwise from top left: An Arduino based plug socket controller, a combined temperature and humidity sensor with ESP8266 controller, a modified off the shelf WiFi relay, and a two-line LCD with ESP8266 controller.

6 CONCLUSIONS

This work represents the initial steps in finding a common language between WoT systems designers and Web developers. We have successfully described how the properties of WoT things can be read and manipulated using CSS(T). It also begins the conversation on how to transition the large number of people working in the Web development space into developing for the WoT. While this approach would never be expected to replace interactions with the WoT, we hope it will represent a way of viewing and interacting with it which is compatible with the workflows and paradigms Web developers are used to.

The use of the DOM opens the WoT to the countless JavaScript libraries that are in use on the Web today. This is both an aid to transition and could encourage reuse of libraries over re-engineering new solutions. For example, existing libraries such as jQuery or Underscore could be used to perform actions on the DOM of an environment. Also, scripts and daemons can be written in JavaScript and run on a Node.js server to execute actions at certain times, under certain conditions or as a response to actions from another device or service.

We have not yet touched upon the more advanced features of CSS such as cascading, conflict resolution and inheritance of property values, which are still open problems for us. We hope they would follow similar, if not the same, rules as CSS does on the Web.

Inheritance of values is a feature in particular in need of discussion. Some may expect an oven's heating element to be turned off when the parent element, the oven, is given an 'off' signal, but few would expect the oven's clock to turn off when the same command is given. However, structurally they appear the same in a DOM. We are confident that our approach will be flexible enough to lead to solutions to these issues.

7 FUTURE WORK

We see potential for using our work in the production of tools which could further enable Web developers to create code for the WoT using their existing approaches. An IDE could be produced which could detect the current things in the environment and offer suggestions for available CSS(T) properties. It could also read actions and events and suggest completions for JavaScript functions and event watchers. Linked data attached to the things and their TDs could enrich the IDE with further relevant information as, for example, the name of a property alone may not be enough to determine how it will affect the thing.

Developer tools for the WoT could use the same technologies to connect to accessible things and allow the environment to be viewed, inspected and manipulated as a DOM, similar to the inspector feature present in modern Web browsers.

The concept of creating content for physical environments is also an exciting one, as this is a large industry on the Web with the potential to expand into the physical world. Creation of CSS(T) represents an initial step to create reusable content for the WoT. Where we have WordPress themes, we could soon have house themes using CSS(T). Where we have corporate interior design guidelines we could soon match them with corporate CSS(T) style sheets to ensure brand

consistency across physical spaces. The ability to distil the essence of a real-world environment to a set of reproducible rules will hopefully provide a lot of interesting and useful benefits.

We concede that end users, designers and professional content producers could not be expected to build style sheets as they often work at a higher level of abstraction and from a different point of view to developers. We would like to see the production of tools that allow automatic visual representations of things to be generated from a DOM which then allow users to quickly and simply design their chosen environment and send it either directly to an environment, to the owner of a venue, share it with their followers or grant access to any other interested parties.

To further rich content generation, we believe the DOM generated by our approach could be integrated with existing HTML pages to form a hybrid document with scripts and styling that control both the document the user is browsing and their current environment. This has the potential to create immersive experiences for users browsing the Web. However, we acknowledge there are significant security issues to overcome before this is feasible.

We have not yet explored the potential for CSS(T) equivalents for CSS animations and transitions, which could provide simple features such as fading lights and gently controlling temperature changes. Advanced CSS features such as these could be implemented in future versions.

ACKNOWLEDGMENTS

We acknowledge the support of EPSRC and the Web Science Institute.

REFERENCES

[1] Arnal Dayaratna. 2017. IDC's worldwide developer estimate, 2016: Emerging geo-developers play a key role. Retrieved January 5, 2019 from https://www.idc.com/getdoc.jsp?containerId=US43033317

[2] Otto Hylli, Anna Ruokonen, Niko Mäkitalo, and Kari Systä. 2016. Orchestrating the Internet of Things Dynamically. Proceedings of the 1st International Workshop on Mashups of Things and APIs. DOI: https://doi.org/10.1145/3007203.3007216

[3] Sebastian Kaebisch and Takuki Kamiya. 2018. Web of Things (WoT) Thing Description. Retrieved January 5, 2019 from https://www.w3.org/TR/wot-thing-description/

[4] Robert Kleinfeld, Stephan Steglich, Lukasz Radziwonowicz, and Charalampos Doukas. 2014. glue.things - A mashup platform for wiring the internet of things with the internet of services. WoT '14 Proceedings of the 5th International Workshop on Web of Things, 16-21. DOI: https://doi.org/10.1145/2684432.2684436.

[5] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. 2012. Internet of things: Vision, applications and research challenges. Ad Hoc Networks, 10, 1497–1516. DOI: https://doi.org/10.1016/j.adhoc.2012.02.016

[6] Martin Schuhfuss. 2015. Let there be light!. JSConf 2015. Retrieved January 4, 2019 from http://slides.com/martinschuhfuss/let-there-be-light#/6/16

[7] Lea Verou. 2015. Proposal: Generalized cascading sheets. 2015. W3C Houdini Public Mailing List. Retrieved January 6, 2019 from https://lists.w3.org/Archives/Public/public-houdini/2015Jul/0003.html

[8] W3C Web of Things Working Group Charter. Retrieved January 5, 2019 from https://www.acm.org /proceedings-template

[9] W3C Web of Things Working Group Homepage. Retrieved January 5, 2019 from https://www.w3.org/WoT/WG/