UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
Web and Internet Science

Applying Web Technologies to Large Scale IoT Deployments

by

Alex Owen

MEng, MSc

ORCiD: 0000-0001-9152-8932

A thesis for the degree of
Doctor of Philosophy

August 2022

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
Web and Internet Science

Doctor of Philosophy

Applying Web Technologies
to Large Scale IoT Deployments

by Alex Owen

This thesis investigates the potential for using Web technologies, namely the Document Object Model (DOM), Cascading Style Sheets (CSS), and JavaScript, to describe and control large-scale Internet of Things (IoT) deployments. These are not yet present in the typical home, however, the trajectory of integration between everyday objects and computers suggests that, in future, many homes and commercial spaces will contain thousands of IoT devices. These environments will require complex and scalable orchestration. Web technologies could fulfil these requirements.

While there have been many attempts to integrate the IoT with the Web, thus far none have taken advantage of existing technologies to the degree demonstrated here. Several new approaches were explored, each with the aim of representing IoT devices and their components using the DOM, whereafter, CSS was used both to control and store the state of the DOM. The DOM elements became digital twins of the devices they represented, allowing actions upon the DOM to be replicated across the IoT environment it mirrors. Through applying this approach, there is the potential for Web developers to use their existing skills to transition from their current role and become Web of Things (WoT) developers with little effort.

The investigation occurred across four experiments, approaching a Web-native solution. The final implementation was tested in a study with experienced Web developers, yielding a positive outcome. Participants showed an interest in the subject matter and quickly learned the skills necessary to implement the technology. Also explored are ideas on how this approach could, in future, be integrated with the social machine of the Web, including with other WoT projects, development communities, and end users.

The proposals within this thesis introduce a new concept for modelling the IoT, and, as such, they put forth many avenues for future research. These include the potential to share curated themes for physical environments; to build complex virtual devices from the components of others; and to allow Web pages to spill out into the physical environment they are viewed within.

Contents

List of Figures

List of Tables

Declaration of Authorship

Acknowledgements

I wish to thank my lead supervisor Kirk Martinez for being persistently supportive of my work and for being on my side through the personal challenges I overcame during the course of this project. His positivity and optimism gave me hope when I was less than hopeful, and the many chats we had gave me a wealth of knowledge about the field. I would also like to express my appreciation for my second supervisor, Erich Graf, who helped me to find the direction I wanted to take during the initial stages of my course.

Additionally, the members of the faculty postgraduate office helped more than they likely realised, by providing breathing space and navigating the university system when I was not able to.

My study would not have been possible without the help of the many participants who gave up their valuable time for the slim chance of winning a voucher. Their combined hours of work gave validity and context to my own.

I would also like to express my gratitude for the support given by Eva, Sami and Spencer at various times throughout my project. It would have been a different and far less enjoyable experience without them.

Finally, thank you to my parents, for supporting my interest in computers both financially and with their time and patience. As well as to my Grandad, for showing me how to program in BASIC for the first time, which is one of my earliest memories.

With the oversight of my main supervisor, editorial advice has been sought. No changes of intellectual content were made as a result of this advice.

Abbreviations

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
API	Application Programming Interface
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
AWS	Amazon Web Services
AXR	Arbitrary XML Rendering
CHSS	Cascading HTML Style Sheets
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DCS	Distributed Control System
DoD	Department of Defense
DOM	Document Object Model
DRY	Don't Repeat Yourself
DSSSL	Document Style Semantics and Specification Language
FOSI	Formatting Output Specification Instance
HTML	HyperText Markup Language
IEEE	The Institute of Electrical and Electronics Engineers
IFTTT	If This Then That
IIoT	Industrial IoT
IoT	Internet of Things
IOTML	IoT Markup Language
IOTSS	IoT Style Sheets
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JS	JavaScript
JSON	JavaScript Object Notation
MAC Address	Media Access Control Address
MQTT	Message Queuing Telemetry Transport
NAT	Network Address Translation
OSI Model	Open Standards Interconnect Model
Project CHIP	Project Connected Home over IP

PWP	Pei Wei’s Proposal
QR Code	Quick Response Code
RRP	Robert Raisch’s Proposal
SCADA	Supervisory Control And Data Acquisition
SHP	Steve Heaney’s Proposal
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TD	Thing Description
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
WoT	Web of Things
WoTIG	WoT Interest Group
XSL	Extensible Stylesheet Language
XSL-FO	XSL Formatting Objects
XSLT	XSL Transformations

Chapter 1

Introduction

This thesis describes the journey towards a standards-compliant approach to integrating the Internet of Things (IoT) with the Web; the merger of these being commonly referred to as the Web of Things (WoT). This work goes on to explore whether the approach put forth for realising the WoT could be deemed to be both technically rational, and helpful for the development community. What sets this thesis apart from other research and commercial implementations is the idea that IoT devices could become an integral part of the fabric of the Web rather than merely a service called upon by users. Such tight integration could enable information to flow beyond the confines of the screen to become represented within the user's physical space, wherein their physical devices become a dynamic part of the content they are viewing. It is hoped that this can be achieved by working within the existing landscape of the Web and, in so doing, would result in far less resistance from the community than a more radical approach.

The questions this thesis sets out to explore are if it is feasible to build a WoT system from this perspective; whether it is practical and efficient; and whether the community would embrace or reject such a system. While these questions cannot all be answered unreservedly, they nonetheless provide a solid basis for exploring the potential of the concepts described herein.

Three variations of a technical approach are explored, all of which centre around using extant Web technologies to solve the relatively new problem of integrating large numbers of IoT devices with other elements of the pre-existing infrastructure. Re-use, expansion and evolution of existing specifications is a core aspect explaining how the Web has grown to the size it is today and thus it is logical to explore this avenue before seeking to create a new technology to solve this problem. When this project started there had been no previous attempts at such a close integration between the IoT and Web (while using Web standards) and even now, none have progressed beyond superficial implementations [1] [2] [3].

The front-end, or user-facing part, of the Web is built almost entirely upon three standards, namely: HTML [4] (describing document structure); CSS [5] (a group of specifications defining presentation); and JavaScript (an implementation of ECMAScript [6]). Together, they allow structured data and text to be presented to the user in a functional, pleasing and interactive manner. At the simplest level, HTML contains data and text in a machine-readable format, while CSS determines how it is presented to the user using a rules-based engine, and JavaScript provides a programming layer that renders the content both dynamic and easier to navigate. Each is an evolving standard, but all have been in mainstream use since the mid-1990s. So far these technologies have underpinned trillions of user interactions with blogs, online retailers, information kiosks, mobile phone applications, refrigerators and cars, inter alia. Using them for applications beyond the confines of the documents they were originally created for is not in and of itself novel [7] [8], although embedding physical devices within documents is entirely new.

By pushing the limits of HTML and the Document Object Model (DOM) [9], it relies on, digital twins of IoT devices can be represented as elements within the document. These twins and the devices they represent can be treated as having equal status, so that a command sent to one will affect both. For example, a 'power on' command sent to a lamp would be received by both the twin and device or else propagated by one to the other, such that both change state to 'on'. Although the state may be represented differently by each, both hold equal precedence and, in any practical way, both are the same lamp. This enables the placement of these digital twin elements into the hierarchical structure of the DOM, alongside other HTML elements containing text and data. Once these physical devices are embedded within the document, they can be manipulated in the same way as any other element of the document, in that CSS can be applied to them, and JavaScript can manipulate them. This thesis tells the story of arriving at this juncture, and discusses where it could, and should, go beyond this point.

Exploring the space involved progressive steps from the genesis of the idea through to failures and technological advancements. This line of enquiry eventually resulted in a system that is nearly compliant with Web standards, and can control IoT environments of nearly any size. The initial experiment was not a practical success, as the approach would have only allowed the use of the DOM and some aspects of JavaScript; however, its XML-based implementation proved to be the seed for the remainder of the project. While it was abandoned early on, it is included as a background, underpinning the later experiments. The second and third experiments grow closer to alignment with how a Web browser manages a Web document and reveal how the project grew alongside the development of the browsers they were based within. The second experiment is a fully functional, yet flawed, IoT system using HTML, CSS and JavaScript; while the third is a far less flawed, yet altogether more superficial simulation of an IoT environment that demonstrates how far this approach could go. The system built for the third experiment embeds IoT devices within the DOM and makes them almost indistinguishable from other elements therein, and thus serves as the most successful aspect of this project.

Measurement of success is not trivial, as there are no systems to directly compare against. Most IoT systems are either proprietary commercial ventures with closed source; open source integrations between closed platforms; or academic systems designed to demonstrate a particular feature or effect. The systems built for this project fall into the latter category and, therefore, any comparisons of performance or completeness would be unfavourable and thus ultimately pointless. Instead, success was measured based on the parameters of possibility, practicality and acceptability. Possibility, which addresses whether an idea can be built into a functional system, is almost always positive, yet comes with nuances. Practicality and acceptability are much more complex and frequently only addressed via subjective measurements. Practicality attempts to answer the question of whether an idea should be used, while acceptability extends this by investigating whether it would be a sufficiently good solution for those who would use it. The fourth experiment delves deepest into this last element by engaging the community with testing and direct feedback.

Overall, the results were very positive. The first experiment was abandoned because it was ultimately impractical, while the second and third were both deemed practical solutions to differing degrees. The system of the third experiment was the only one that was considered viable for testing by those other than the author and this was done so in the fourth experiment. The fourth experiment received a great deal of constructive positive feedback and suggestions, albeit with the caveat that the participants were pre-filtered by both selection bias and survivorship bias. However, the results suggest that a significant proportion of Web developers, a group suggested to be around 24 million globally [10], would be able to use the system and would find it intuitive and enjoyable.

Community acceptance is of substantial importance as developers are under great pressure to keep up with the pace of Web technology which entails constantly learning new frameworks, tools, and applications (and new iterations of each). Current development for the IoT is a fragmented landscape of proprietary systems and APIs, and even the newer approaches of Matter [11] and the W3C WoT Group [12] are complex and require learning new mental models and syntaxes. If IoT development on the Web (WoT development) were a progression from Web development, this would mean that developers could build on what they already know, using their pre-existing knowledge to occupy the new niche of the WoT. Thus they would not need to treat it as a new technology and it would not require them to learn new concepts from scratch. This could mean faster adoption of the WoT or else a decreased mental burden for those choosing to adopt it.

To provide context to this problem, this thesis first explores the background to the WoT within the literature review. This addresses the history and evolution of the Internet and the Web and compares their development with that of the IoT and WoT. These two progressions follow many of the same milestones, which may indicate that a similar approach to problem-solving can be applied to both. Once the background is established, the theoretical framework chapter outlines the potential use of Web standards to resolve issues in the WoT, defines the niche that this thesis intends to fill, and lists the aims it will explore. This chapter also describes the approaches for determining the possibility, practicality and acceptance of each experiment.

The subsequent chapter details the four experiments carried out in pursuit of a WoT system based on Web standards. The experiments are progressive and each builds heavily on the previous one. Each experiment is self-contained and, within it, are the methodology, results, and discussion. The ensuing discussion chapter draws these four experiments together and looks into the implications of what was achieved and conveyed.

The conclusions chapter takes a higher perspective of what has been produced and discovered and examines how it sits within the wider context of the Web, including the social and technical structures it affects. This final chapter concludes by addressing future work which could be carried out using the ideas presented in this thesis. It provides a list of potential applications for this implementation of the WoT alongside other technical and social considerations that the author feels should be explored.

Chapter 2

Literature Review

2.1 The Internet

The precursor to the modern Internet was the Advanced Research Projects Agency Network, or ARPANET. It was a long-distance network with only a few nodes located within those organisations which assisted in its development. When the ARPANET was first built in 1969 [13], it was simply a connection between the UCLA and SRI International at Menlo Park, California. Later that same year, UCSB and UTAH were added to the network. In 1970, a connection was extended from the west coast to the east coast of the USA, linking MIT, BBN and Harvard. By 1973 the network had grown to around 40 nodes at different locations, integrating approximately 45 computers. These computers were room-filling mainframes and would be almost unrecognisable to today's average user. Also in mid-1973, a connection was made to University College London via satellite link, as seen in Figure 2.1, establishing it as an international network for the first time. Over the first few years, the network continued to grow, serving as a bridge between many smaller academic and military networks and, by 1977, many other institutions had been incorporated, as seen in Figure 2.2. The growth of the ARPANET was steady and rapid and it was very soon impossible to graph the entire network due to its size, rate of expansion, and shifting topology. Other smaller networks rose alongside the ARPANET and these were all slowly linked together, eventually forming one larger network which has since become what we now call the Internet. Today the Internet is a planet-wide network connecting billions of devices and, as such, has become the basis of almost all modern communication. Connected devices vary from warehouse filling supercomputers to smartphones as well as smaller embedded computers inside other objects.

The core of the Internet is machine-to-machine communication and the continued existence of the Internet relies on devices being able to communicate with one another using a shared language or, at the very least, a set of languages that can be translated automatically. This interoperability is largely upheld by shared standards and protocols

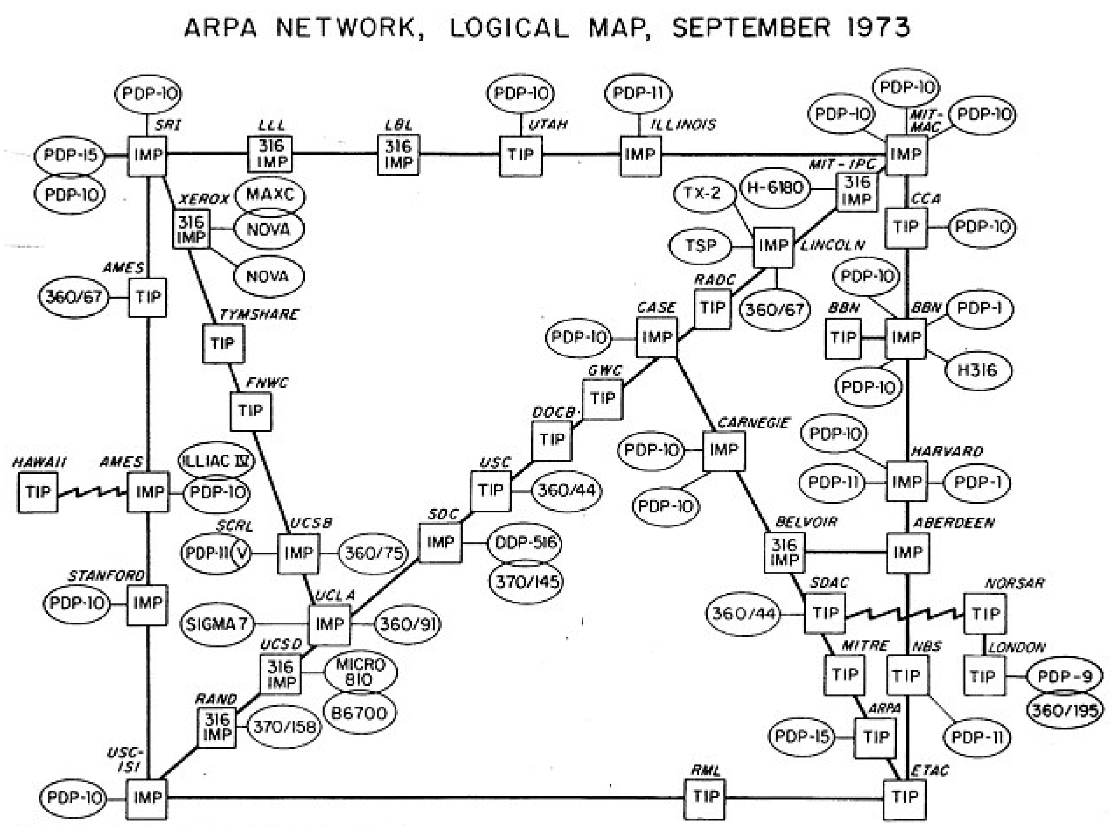


FIGURE 2.1: A schematic of the ARPANET in late 1973, from DARPA's 1978 report [13].

such as the Internet Protocol Suite [14], which was proposed in 1974 [15] to support the ARPANET. This collection of protocols allows individual devices to find paths to one another and to send messages in an agreed format which can be reconstructed and read upon arrival. The Internet Protocol Suite is arranged into a series of layers which each perform functions at a different levels of abstraction.

The Internet Protocol Suite

The constituents of the Internet Protocol Suite are Transmission Control Protocol (TCP), which handles ordering and error checking of streams of packets, and Internet Protocol (IP), collectively called TCP/IP [16] which routes packets from their source to destination based the packets' headers and unique addresses. Alternatively to TCP, devices can use UDP (User Datagram Protocol) which is a simpler and sometimes faster protocol, but one that lacks an error checking capability.

IPv4 has been prevalent for much of the life of the Internet. However, the scale of the connections and the need for direct connectivity to devices (rather than hiding them behind routers) led to the development and implementation of IPv6. It was first proposed in 1998 [17] and subsequently revised in 2017 [18]. IPv6 is a key enabler for the IoT as the IoT brings many more devices into the network, and IPv6 has a

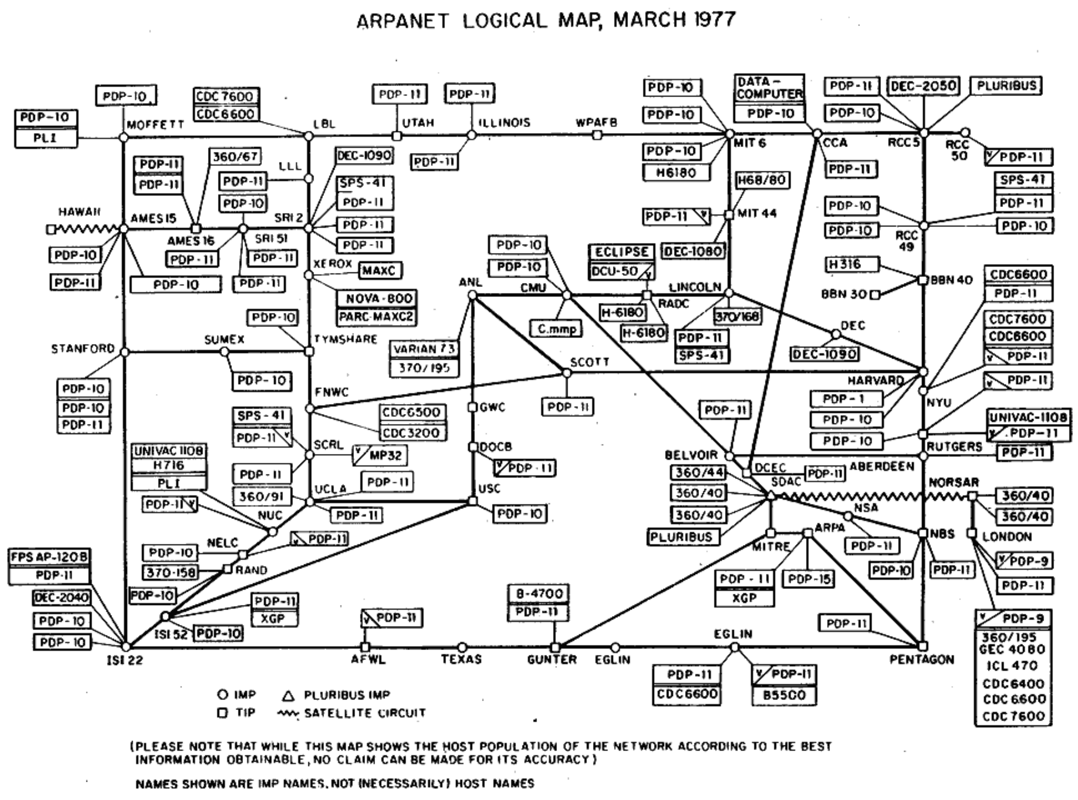

ARPANET LOGICAL MAP, MARCH 1977
O IMP
□ TIP
△ PLURIBUS IMP
SATELLITE CIRCUIT
(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)
NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

FIGURE 2.2: A schematic of the ARPANET in 1977, from DARPA's 1978 report [13].

much larger address space than IPv4. Without it, very large-scale IoT deployments would be difficult or impossible.

However, machines are not the only part of the Internet. While there is a lot of machine-to-machine communication, there is a very large human component of the Internet, creating a highly complex sociotechnical machine. This is particularly evident as we move towards the realisation of the IoT [19]. People use the Internet to communicate indirectly with one another, but also provide data and close interaction loops between systems that have no programmatic link. Given the Internet is formed of both machines and people, at the lowest level of complexity, interactions can come in three formats: between human and machine, machine and machine, and human-to-human communication, as seen in Figure 2.3.

Human-to-human communication can arise directly within this social machine, for example by asking someone to look up some information as in Figure 2.4. It can also emerge indirectly in a chain of interaction types comprising of human-to-machine communication, followed by machine-to-machine, and ending with machine-to-human. Such communication can also manifest in both real-time or delayed (asynchronous) forms. Voice and video messaging can be almost real-time, although some services allow for messages to be recorded and picked up later. Text chat can be real-time or asynchronous depending on

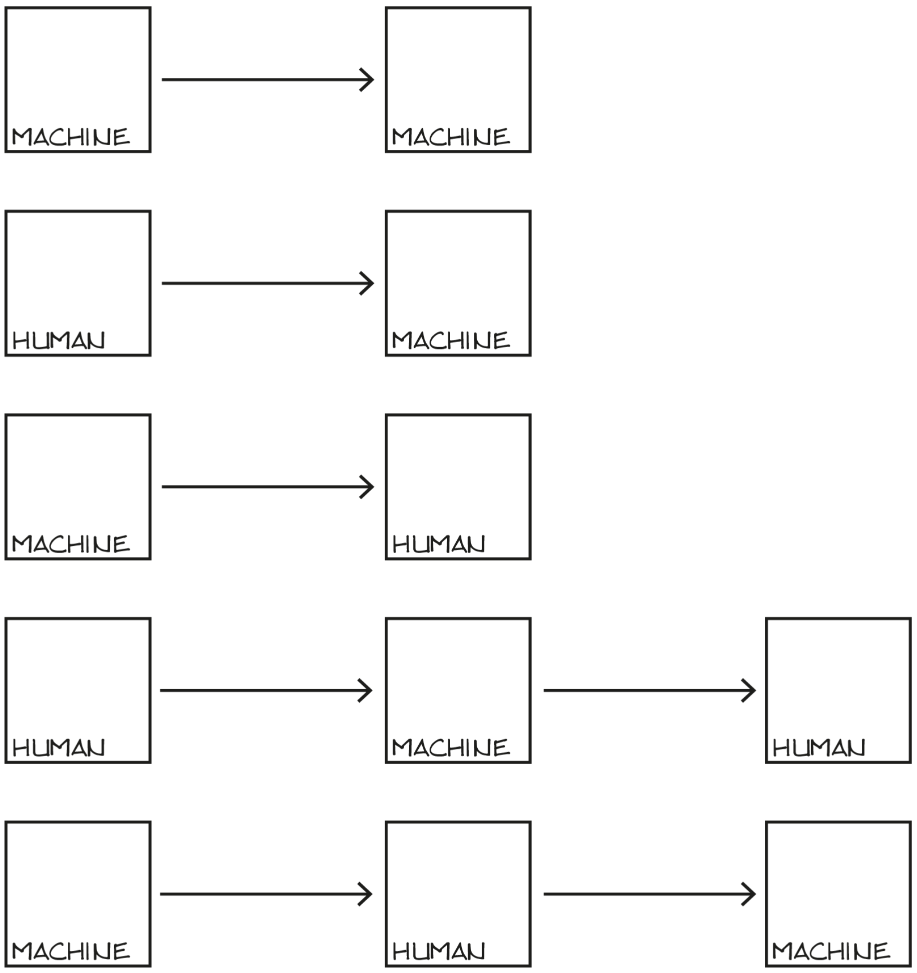


FIGURE 2.3: Communication types as used within the Internet and IoT.

whether all parties are online at the same time. Many forms of communication can involve two parties, yet also allow for many more to join. It can be multi-directional, such as in a conference call, or unidirectional, as a broadcast message. Despite having three relatively simple building blocks, the machine as a whole can become very complex.

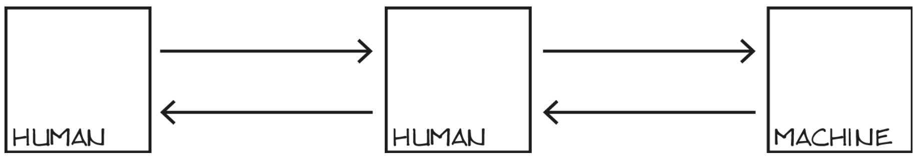


FIGURE 2.4: Asking someone to look up some information using the Internet.

Human-to-machine and machine-to-human communication will often be concurrent as the human will anticipate some form of feedback from the machine they have commanded or queried. However, they can be seen alone, such as in an automated, scheduled message from a weather service to a user. Such communication can be either explicit or implicit. An example of explicit communication would be a user clicking a button to request traffic information, but they could also receive this information implicitly by arriving at a certain location. Such implicit communication blurs the line with machine-to-machine communication as, while the user would have given permission for their smartphone to track their location and send it to a traffic server, they have not actually sent that location message themselves, rather, they have given agency to their smartphone and delegated a task.

Machine-to-machine communication is much more common on the Internet, as every indirect communication that arises between humans also involves many messages transferred between machines. In the case of audio and video, this could amount to many thousands of messages per second. There are also autonomous machines communicating with other autonomous machines, for instance a rain sensor sending messages to a window to let it know it needs to close; or a robot on a production line sending alerts when it needs maintenance.

Various forms of communication are extremely important in a world where there are growing numbers of computers embedded within everyday objects causing us to increasingly delegate agency to automated systems. The Internet and the Web are the basis for taking this agency beyond the confines of the device with which we interact, further delegating it to remote servers or actuating remote systems. The needs and desires for people to do this have driven many technical advances surrounding User Experience (UX) and Human Factors research. Currently, the Web is the user-facing part of many of these interactions, which may imply that the WoT could become another aspect of this interface.

2.2 The Web

2.2.1 Precursors to the Web

The concept of hypertext has a history that dates as far back as “The Garden of Forking Paths”, a story that was written in 1941 by Jorge Luis Borges [20] and the Memex machine envisioned by Vannevar Bush in 1945 [21]. The first hypertext computer system was Project Xanadu [22] which was created by Ted Nelson in 1960. Nelson also coined the terms ‘hypertext’ and ‘hypermedia’ in or before 1965 [23]. These, and other systems, all served as inspiration for the Web.

2.2.2 The World Wide Web

The World Wide Web is a layer on top of the Internet that was predominantly created for asynchronous, indirect human-to-human communication. The concept of the Web was initially put forward by Tim Berners-Lee while working at CERN in 1989 [24], where he created an internal system for sharing knowledge based on hypertext [25]. However, the limits of the Web have extended far beyond this initial idea and, in modern systems, many machines use Web-based technologies to communicate with one another, both as a part of the human-to-human chain, and completely separate from it.

The Web, when used by a person, is primarily engaged with via a Web browser such as Google Chrome [26], Microsoft Edge [27], Mozilla Firefox [28], Opera [29], or a task-specific browser, such as a messaging application or social media application. Regardless of their specificity, an item of Web software must have an implicit understanding of the protocols necessary to communicate with Web servers so that it can both send requests to the server and decode and present the responses it receives. This, in turn, implies that it must also be able to use the Internet Protocol Suite upon which the Web is constructed. The key Web protocols include HTTP and FTP, while the technologies required to decode the data include parsers for HTML, XML, CSS and JavaScript. HTML thus provides the structure of the data and CSS the presentation, while JavaScript allows for interactivity between the user and the data. HTML and the document structure of the Web are foundational to this thesis. A Web page stores information as a hierarchy of nodes containing text and images and, more recently, videos, embedded applications and much more. This structure came to be standardised as the Document Object Model, or DOM [9].

Read/Write vs. Read-Only

Initially, the Web was designed to be both read and write [25]. Web pages allowed users to edit the page's content and replace what was already there. This was later changed so that users who did not control the server could only read a page's content. It was not until the era of Wikipedia [30] and Web 2.0 in the early 2000s that this concept returned, and then it usually came with strict permissions structures to prevent misuse. Wikipedia initially allowed any user to edit a page, but after several disingenuous edits which harmed the site's reputation for accuracy, a permissions and approval process was put in place. This was accompanied by a hierarchy of editors who monitor high profile pages. In a 2012 study commissioned by Wikimedia (Wikipedia's parent company), an independent review found its accuracy to be extremely high [31]. While the consensus was that it is not suitable for citing in most academic contexts, it has since achieved a level where it is generally equivalent to any other tertiary source and is thereby useful for referencing concepts and definitions. This marks a milestone for the read/write Web and demonstrates that collaboration can lead to a positive outcome.

The IoT follows a similar read-only and read/write structure to the modern Web, excepting that devices join humans as actors within the system in that they can read from, or write to databases and interfaces, provided they have been explicitly or implicitly granted permission. Devices and humans can also write new states to devices, such as updating the volume of a speaker. In this way, parallels can be drawn between a document on the Web and a device within the IoT/WoT.

The Web is defined by an ever-changing group of specifications that have grown both in number and complexity, with new features constantly added by its guardians, the World Wide Web Consortium (W3C). One particularly interesting and relevant recent development is that of Web Components. This is a group of four specifications which combine to allow the user to define a new element of a Web page with a specific function, much like a `<div>` or `<header>`, but with a purpose as defined by the developer. These specifications are delineated as Custom Elements [32]; the Shadow DOM [33]; the HTML Template Element [34]; and ES Modules [35]. These four combined enable Web Components to function and, while it is not self-contained, for brevity they will henceforth collectively be referred to as the Web Components specification. This specification allows the developer to create new elements by creating a new type of DOM object that can be inserted into the DOM tree and processed at runtime in the same way as an HTML element. The aim of the specification is to allow for easy re-use of complex custom components, much like Angular [36], React [37] or Vue [38] have done. For example, a developer may create a component for a form, a button, or a video player. However, the specification is deliberately unconstrained and this new element could semantically represent anything, including a digital twin for a physical device, a concept that will be explored in much more detail from Chapter 3 onwards.

In addition to the W3C, the Web Hypertext Application Technology Working Group (WHATWG) is another group which develops standards for HTML. They were formed after a W3C workshop in 2004 [39] as an alternative forum for discussion as to how to progress the language after growing frustration that the W3C was not paying due attention to it. The founding members were Apple, Mozilla and Opera, although it was intended to be an open community, given that anyone can submit a proposal to their GitHub repositories [40] which hold the standards. The membership and ideas of the two groups often overlap, with the WHATWG being more commercially-focused than the W3C.

2.2.2.1 HTML and the DOM

HTML contains the structure of a Web page in the form of tags that are nested within one another, each representing an element within the DOM tree. It was typically used to mirror paper documents at the origin of the Web, but has since expanded to include semantically marked up audio, video, code in various languages, and many others. Each element has a semantic meaning. For example, this may include a paragraph, heading, or article. The DOM is the model used to hold the data which can be represented in HTML format [4]. It is a tree structure wherein each node is an HTML element and each is either a child of the root node or else of another HTML element.

HTML elements can have attributes that describe the state of the element, such as `disabled`, `type` or `value`, or provide implicit links to other resources, such as the location of a given CSS file, media file, or script. While these are fixed for the core HTML elements, Custom Elements and XML elements are not restricted to a list of allowed attributes. This approach of linking state to HTML elements is used extensively in Experiment 1.

SGML, XML and XHTML

SGML [41] is a meta language designed to define markup languages, including HTML and XML. Both HTML and XML were intended to be implementations of SGML despite HTML being created in parallel to SGML.

XML [42] was created as a more generic version of HTML [43] with the ability to add arbitrary tag names and attributes. It is mostly used to represent data in a similar way to JSON. XML parsers tend to be deliberately less fault-tolerant as a method to encourage better-formed data. Rather than throwing an error and continuing, they will usually fail the entire process. A key difference between XML and HTML is that every tag must be closed.

XHTML [44] is a version of HTML represented as XML. The differences to HTML are slight, and mostly around including closing tags, but the result is a markup that can be rendered as HTML but processed as XML. A key benefit the creators saw was modularisation [45], which would have allowed sections of reusable XHTML to be defined with XML schemas. However, this never materialised as a need within the development community, especially after the advent of component-based frameworks.

While the DOM structures data, its presentation to the user and interaction are administered by CSS and JavaScript, respectively. These handle how the contents and state of the data are presented to the user and enable the user to alter the state locally, as well as to communicate alterations to the server if required. CSS provides set-based rules which can determine how the DOM is displayed, but beyond that, it contains a language for describing the state of nodes in a tree. JavaScript, amongst its many other features, offers the potential to integrate other data sources with the data in the DOM to present a unified experience, or else to produced derived data. Ultimately, it can build a DOM on the fly, which is not directly sourced from an HTML file, yet can still be represented as HTML due to the two-way relationship arising between the DOM and HTML.

2.2.2.2 CSS

For a time, the Web simply comprised text and basic images wherein each page was treated as a digital data store. It was not until later that presentation of that data started to become more important. The direct precursor to CSS, CHSS was first proposed in 1994 by Håkon Wium Lie [46] and was one of several attempts to standardise a method for describing the layout and style of a document. Initially, it aimed to replicate typeset documents in a digital format but, as the Web has grown exponentially, it has since expanded to contain much more. While it originally implemented static designs, it now contains animations and transitions, within an ever-growing feature set.

CSS is designed to work in conjunction with HTML and uses unique identifiers known as 'id' properties, and classes, defined in terms of 'class' properties, attached to HTML elements. Each element can have multiple classes but only one identifier, and an identifier should be unique within a document. This allows CSS to define presentation rules that apply to individual nodes or sets of nodes, thereby allowing union, intersection and complement operations to be applied with relative ease.

CSS and Set Theory

CSS is designed to select both individual elements and sets of elements. It has a variety of operators that allow for coverage of basic set theory concepts.

An individual element can be selected with the `#` operator. For example, `#switch` selects an element with an id of 'switch' (e.g. `<button id="switch"></button>`).

An entire class of elements can be selected with the `.` operator. For example, `.text` selects all elements with a class of 'text' (e.g. `<p class="text other-class"> </p>`).

Union:

A union can be described using the `,` operator: Thus `.button, .carousel` would select all elements with a class of 'button' or a class of 'carousel', including those that have both: `button` ∪ `carousel`, or logical OR.

Intersection:

An intersection can be achieved by not leaving a space between selectors: Thus `.carousel.images` would select those elements which have both a class of 'carousel' and 'images': `carousel` ∩ `images`, a logical AND.

Absolute Complement:

The absolute complement is found with the `:not()` operator, where `:not(.banner)` selects all elements except those with a class of 'banner', a logical NOT.

Relative Complement/Set Difference:

The relative complement can be selected using the `:not()` selector. `.carousel:not(.images)` would select elements which have a class of 'carousel' but not those that have a class of 'images'. If an element has both, it is also excluded: `carousel \ images`, which does not have an equivalent logical operator.

Symmetric Difference/Disjunctive Union:

The symmetric difference also uses the `:not()` selector in a similar way to the relative complement: `.carousel:not(.images), .images:not(.carousel)` is a little verbose but achieves the goal of selecting all elements that have either the class of 'carousel' or 'images', but not both: `carousel` △ `images`, a logical XOR.

The CSS Cascade

Cascading is a feature of CSS that sets it apart from many other approaches. The cascade defines the order that rules are applied to the document, based upon a pre-determined hierarchy of the importance of these rules. The implementation of which allows multiple style sheets to be imported and combined without earlier imports being overwritten by later ones. While the order in which they are imported into the document provides a reconciliation for two equally important rules, relative importance based on rule type is far more critical. The CSS cascade specification [47] lays out the precedence for these rules and includes deciders such as rule specificity and source.

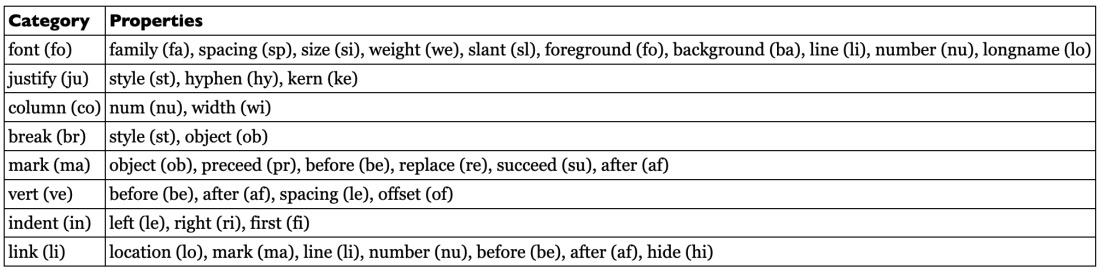
2.2.2.3 Precursors and Parallels to Web-based Style Sheets

The following two sections (see also Section 2.2.2.4) are heavily influenced by Håkon Wium Lie's PhD [7], as it is useful as both a primary and secondary source, and is laid out in an appropriate structure for this thesis. The standards and proposals outlined in this section are those which arose between the late 1980s and mid-1990s and had an influence on both the creation and ongoing development of CSS. There were many proposals for how to style documents and, while many contributed directly or indirectly to the final CSS concept as developed by Håkon Wium Lie and Bert Bos, some were also developed in parallel and influenced the standard much later on. CSS was adopted by the W3C as the preferred approach for HTML and, as a result, has the largest adoption of all those in the group of style sheet specifications.

FOSI Formatting Output Specification Instance (FOSI) [48] was a style sheet language initially created by the US Department of Defense (DoD) and posted to the www-talk mailing list and comp.infosystems.www newsgroup. It was written for SGML and was later adapted specifically for XML. It is itself written in SGML, similarly to Document Style Semantics and Specification Language (DSSSL), and was only ever intended as an interim solution [49] which was designed to fill a need while DSSSL was being developed. It was used in the period between the codification of SGML by the ISO in 1987 and the release of DSSSL. However, it was never widely used outside of the DoD, nor was it intended to be, yet it was an important step in the progress of styling structured data.

Mosaic: RRP RRP was a proposal by Robert Raisch (of O'Reilly) and subsequently referred to by those involved as Robert Raisch's Proposal. It was an RFC made to the www-talk mailing list for "an easily parsable format to deliver stylistic information along with Web documents" [50]. Up until this point, browsers presented the content in the way they thought best, which often led to inconsistent results. This was not helped by the mix of text-based and graphical browsers at the time. The RFC goes on to iterate and reiterate that the "stylistic information" should be considered only as a suggestion, and not as a "required behaviour", an ethos that is contrary to modern CSS.

The syntax is different from CSS and was designed to be very compact, as connection speeds were slow and GZIP [51] was not yet available. Specifically, this choice aimed to "minimize the time required to retrieve and interpret" the style. The specification lacks many features of CSS, yet the concepts are superficially similar. It used the @ symbol to select elements by their name, but did not allow for selection based on attributes or allow concurrent use of selectors in a single rule. It also had a set of 35 properties, grouped into eight categories, as shown in Figure 2.5 from Håkon Wium Lie's PhD thesis [7].



Category	**Properties**
font (fo)	family (fa), spacing (sp), size (si), weight (we), slant (sl), foreground (fo), background (ba), line (li), number (nu), longname (lo)
justify (ju)	style (st), hyphen (hy), kern (ke)
column (co)	num (nu), width (wi)
break (br)	style (st), object (ob)
mark (ma)	object (ob), preceed (pr), before (be), replace (re), succeed (su), after (af)
vert (ve)	before (be), after (af), spacing (le), offset (of)
indent (in)	left (le), right (ri), first (fi)
link (li)	location (lo), mark (ma), line (li), number (nu), before (be), after (af), hide (hi)

FIGURE 2.5: Table 7 taken from Håkon Wium Lie's PhD thesis [7].

For example, `@H1 fo(si=32,we=bo)` would set the font (`fo`) for an H1 element to a size of 32pt (`si=32`) and give it a bold weighting (`we=bo`), which has the equivalent in CSS of:

```
h1 {
  font-size: 32pt;
  font-weight: bold;
}
```

RRP was limited in that it did not have the cascading features of CSS or allow for multiple style sheets in a single document. While Marc Andreessen was aware of RRP [52], it was never added to Mosiac and he instead opted to use presentational HTML. This is where the HTML defines the appearance directly using tags such as `<BLINK>` and `<CENTER>`. Many saw this as a backward step in the process of developing standards and, in 1997, the W3C agreed, publishing a recommendation that style sheets be used over presentational HTML [53]. However, it was only much later that the last remnants of presentational HTML were removed by the HTML5 standard [4] in favour of making HTML more semantically meaningful.

ViolaWWW: PWP Pei Wei's Proposal (PWP) was made on the same www-talk mailing list as RRP. It was designed for the ViolaWWW browser which Pei Wei – also of O'Reilly – created and maintained, and was loosely based on RRP with some significant differences. It used parentheses to allow for multi-level selectors (e.g. `H1` within `BODY`). This allowed for basic `AND` and `OR` functionality, using separate rules at the same level of parentheses for OR and commas for AND. PWP did not abbreviate properties and instead used full names such as `fontSize` and `fontFamily` which are very similar to CSS's `font-size` and `font-family` and otherwise identical to the JavaScript style object of an HTML element which uses camel case. PWP was however not implemented in other browsers and quickly fell out of use.

Steve Heaney's Proposal (SHP) Steve Heaney would appear to have taken objection to PWP as he almost immediately published his own outline [54] for re-using FOSI for style sheets and SGML notation. He felt that it was better to re-use than to create another standard, a sentiment that has been echoed many times in Computer Science. Ultimately, the outline never progressed to a specification as no one took up the task of writing it.

Cascading HTML Style Sheets (CHSS) CHSS [46] is the original language created by Håkon Wium Lie, one that would later evolve into CSS [55]. It was the first to introduce the idea of a cascade, even though the style sheet hierarchy was very different. In this proposal, the suggestion was that users or developers could specify how strong a preference a given style sheet would receive and this would, in turn, decide which rules would be prioritised and to what degree. The syntax of the examples presented in his thesis is very different from that of CSS, and CHSS offers conditional statements that CSS deliberately does not.

DSSSL DSSSL [56] was a language first published as a draft in 1994 and was used to style SGML-compatible documents. It used a subset of Scheme [57] to manipulate SGML documents, altering both their structure and appearance. This is beyond the capabilities of CSS and especially what CSS could do at the time (which was only to alter the appearance of data). Modern CSS pre-processors such as SCSS and LESS, discussed later in this chapter, have adopted some of the capabilities of FOSI and DSSSL as they can perform basic programmatic functions. Ultimately, DSSSL fell out of popularity as SGML has largely been replaced by the HTML and XML standards in modern implementations although DSSSL's successor, XSL, continues to thrive in certain scenarios. Today, DSSSL is rarely used outside of Linux documentation.

2.2.2.4 Modern Alternatives to CSS

XSL Extensible Stylesheet Language (XSL) is a family comprised of three parts [58]; namely: XSLT, XPath and XSL-FO. In combination, these enable customisation of the presentation of XML documents. XSL is the only style sheet language still in widespread use other than CSS. It is based on DSSSL and was an attempt to port that same approach to XML. It is designed to manipulate the structure of XML documents and, as a result, is more complex than CSS. The W3C has long recommended CSS first [59], and suggests only using XSL if its more advanced features are required, such as transforms. This sentiment is shared by Bert Bos (the co-creator of CSS) who said that "XSL is only an alternative at the high-end, for advanced users" in his 2008 essay [60] concerning CSS variables. It has a place in modern Web development although, due to its limited use cases, it would be rare to find a developer who is proficient in it.

The Components of XSL

XSL Transformations (XSLT) allow for "transforming XML documents into other XML documents" [61], including the manipulation and sorting of XML, and by extension XHTML, based content.

XPath is primarily a "means of hierarchic addressing of the nodes in an XML tree" [62]. The latest version supports JSON as well as XML.

XSL Formatting Objects (XSL-FO) are designed to help an "XSL transformation into a tangible form for the reader or listener" [63].

SASS, LESS and Pre-Processors SASS/SCSS [64], Stylus [65] and LESS [66] are style sheet languages designed to be processed into CSS and sit at a higher level of abstraction than CSS. They add many extra features which can reduce the length of CSS and hide repetition from the developer, a feature which is often useful when building websites. These languages include some features that are deliberately left out of the initial CSS specifications, including looping, variables and 'if' statements. Some of these features, including variables, have since been added to CSS although in different ways to those used by the higher abstractions. The key difference between CSS and these languages, however, is the use of nested selectors to replace duplication of the same prefix for several consecutive selectors.

CSS-in-JS CSS-in-JSS, later renamed to JSS [67], is an approach that removes the CSS and replaces it with a JSON-based definition of style. This is similar to how using `HTMLElement.style = {}` allows for setting style on an element using only JavaScript objects. Its key feature is that it automatically namespaces sections of CSS which allows for identical selectors in different parts of the code, ultimately resulting in shorter selectors for the developer to have to understand. Similar to SASS and LESS, it can compile to CSS and, in doing so, will automatically create namespaces for the various sections of CSS to prevent conflicts from arising.

AXR Arbitrary XML Rendering (AXR) [68] was a proposal that was made to replace HTML and CSS with XML and HSS, respectively. HSS, an acronym that was never publicly defined, began as a specification for a superset of CSS, yet ended up becoming a replacement that followed many of the same paradigms and used a similar syntax. The ultimate goal was native browser support although, initially at least, the creators had planned to offer a browser plugin for rendering. Functionally, it offered many of the same features of LESS and SCSS, including variables and nested selectors, but had a focus on object orientation and allowed the derivation of properties based on how another element was rendered.

Ultimately, the idea was abandoned around 2013 due to a lack of support in the community. Much of this was likely due to the creators' insistence that the W3C and WHATWG were flawed in that they were too slow and were "made up of companies fighting each other" [69], instead advocating a grassroots movement of developers and designers. Unfortunately, many of the perceived "companies fighting each other" were the very browser vendors through which they intended to implement their new standard and so this conversation was not initiated on the best footing. There were also technical issues with basing the style of one component on another. Particularly, it could require re-rendering the page many times, as each referenced element in a chain is settled and thus loops would be quite likely to arise in complex designs. Overall, while AXR has not had a significant influence on the technology of style sheets, it does serve as an important lesson that working with the standards bodies is far more productive than challenging them.

2.2.2.5 The Evolution of CSS

While the progression from the original CSS standard to CSS3 was fairly linear, progress since then has been deliberately fragmented by the working group. Due to the size of the language, it no longer exists as a versioned progression, rather the language has been split into modules and each of these will be progressed separately. While there is a CSS4 community group [70] at the W3C, the opening statement outlines that it is a name to be used for teaching and promotion and will not be applied to any specifications. Browsers have always taken a rolling approach to the implementation of CSS, with vendor-prefixed versions of properties released initially and further standards-conforming versions later. Developers refer to sites such as caniuse.com [71] and the Mozilla Developer Network [72] to see whether a particular version of a browser will support the features they want to use and then cross reference that with the browsers and those versions their userbase employs.

2.2.2.6 JavaScript

JavaScript, initially called LiveScript, came into existence in 1995 soon after the arrival of CSS while the Web was still growing quickly. JavaScript offered a way to add interactivity to Web pages and allowed the content to change dynamically based on user interactions. It is a weakly typed programming language that is interpreted at runtime rather than being compiled into an executable form. As a result, it can be delivered as text to the Web browser. This fits well with the textual nature of HTML and CSS, and its simplicity, compatibility and timing have helped it to become the dominant programming language for the Web [73].

Over time there have been several programming languages which have coexisted alongside JavaScript, but none thus far has proven as resilient as JavaScript. Java [74], primarily a desktop language, was embeddable within Web pages for a while. However, it required a browser plugin and a working installation of Java on the client computer. Adobe Flash [75] was also used to make Web page embeddable applications, as was Macromedia/Adobe Shockwave [76]. All of these have been since discontinued due to security issues or simply because they have been superseded by new languages like WebAssembly [77]. Furthermore, advances in the JavaScript language have rendered them obsolete. There are also several other languages, including Dart [78] and Rust [79], which compile into JavaScript or WebAssembly. Despite this competition, JavaScript has thrived and is still among the world's most widely used programming languages. One of its defining features is its ability to evolve and expand alongside the needs of the Web.

JavaScript has typically been employed as a front-end language where it can be seen in use on nearly every Web page although, in recent years, it has made a shift into server-side operations, particularly with the rise of Node.js [80], a headless JavaScript interpreter which was released in 2009. There are also embedded JavaScript interpreters, including Espruino

[81], which run on very constrained hardware, albeit not as efficiently as C would run on the same hardware due to overheads in the interpreter and its relative immaturity.

2.2.3 Scale and the Internet of Things

Over the past three decades, both the scale and heterogeneity of Internet-connected devices have increased beyond reliable measurement. It is this variety of devices that has led to the need to differentiate between those devices which are primarily for computation, such as laptops and desktop PCs, and those for which computation and connection to the Internet is a secondary function (e.g. an Internet-connected light bulb). This is just one of many definitions used by a superset of the Internet which we collectively know as the IoT and the term which will be used for the duration of this thesis.

Peripherals vs. IoT Devices

There has to be a line drawn between that which is a discrete device and that which is a peripheral of another device. This line is understandably blurred. Many peripherals are exceptionally complex. For example, an electron microscope attached to a workstation would probably be considered a peripheral, even though it contains technology many times more advanced than the workstation.

For the purposes of this thesis, the line will be drawn using the relationship type. A peripheral can connect to only a single device at any one time, while a device itself is capable of connecting to many other devices. This would mean that a USB camera is defined as a peripheral, while a network camera is a device. This is a simple delineation but one which means that the definition of a device is usually straightforward and the demarcation between devices is clear. This could of course lead to objects that could be both devices and peripherals in different modes, but such cases are relatively rare. There are complexities. For example, the first webcam in the computer lab at the University of Cambridge (Figure 2.6) was a peripheral, yet the could be considered an IoT device if a line is drawn around the computer, camera, and coffee pot. This line of argument, however, quickly strays into the realm of a metaphysical discussion.

2.2.4 The Web of Things

The WoT is to the IoT as the Web is to the Internet; it is a set of principles and technologies which can be used to involve people in the IoT so that they can understand and engage with it. The WoT is strongly related to the Web, and the two share protocols and

FIGURE 2.6: The first webcam, which was used to monitor a coffee pot in the Trojan Room of the computer lab at the University of Cambridge, operated from 1991 to 2001 [82].

implementations. However, several groups are seeking to add new Web standards which could help to solve some of the unique problems that interaction with the IoT presents.

These groups are both for-profit groups and not-for-profit organisations. The key player is the W3C [83], the organisation that leads the way in developing standards for the Web. They attempt to tread the line between the needs of the general user, those of the businesses involved, and building a sustainable, scalable, secure and efficient system. This process takes time and, in the IoT space, several business consortia have formed to accelerate the adoption of interoperable standards. However, this has often resulted in the effective exclusion of those outside the group. These groups, organisations and consortia are mentioned in context throughout this chapter.

There is currently no equivalent of a Web browser for the WoT, although the W3C's WoT Group sees their Scripting API [84] as the foundation of a browser equivalent [85] for the WoT. However, as there are no finalised global standards, most current IoT devices currently employ Web technologies directly through proprietary applications and not via separate WoT technologies.

Of the major browser players (Google Chrome, Apple Safari, Mozilla Firefox and Opera), none are publicly integrating their browser product with IoT devices. Google and Apple are pursuing separate IoT systems although, for the moment, they currently have not publicised any intention to integrate these with their browsers. Mozilla created Mozilla WebThings [86] although, in September 2020 it became simply WebThings and split from its parent company after funding was removed. WebThings comprises a gateway operating system that is designed to run on an embedded device such as a Raspberry Pi or router and a set of libraries for common languages to allow for the easy creation of WoT devices. The concept broadly follows the W3C's approach by including actions, properties and events as seen in the W3C's architecture document [12] and the Thing Description [87]. Opera has not made any overt movements towards the WoT space.

2.3 Identification on the Internet

There are several ways in which a device can be identified on the Internet, and without this the Internet itself would cease to function, as messages could not be delivered efficiently. Within the Open Standards Interconnect (OSI) model [88], a physical interface can be identified using a Media Access Control (MAC) address. MAC addresses are in the EUI-48 format and administered centrally by the IEEE [89] so that no two devices have the same identifier.

Accuracy of the OSI Model

While the OSI model is a useful teaching tool, it is not necessarily strictly followed in practice. However, for the purposes of this thesis it constitutes a sufficient model to illustrate some of the different identification methods present.

At a higher level, interfaces can be identified using IP addresses, each of which can be converted to a MAC address at the time of use employing the Address Resolution Protocol (ARP) [90]. IP addresses are either IPv4 or IPv6, with the latter having a much larger address space and, therefore, is able to address far more devices. The IP address of a device is unique within the network it is a part of and this may be a local network or the Internet as a whole. Traffic is commonly routed between networks using Network Address Translation (NAT) which allows for the routing of traffic between two devices with IP addresses that are not globally unique.

MAC addresses operate at the Data Link Layer (Layer 2) of the OSI model, while IP addresses operate at the Network Layer (Layer 3). Uniform Resource Identifiers (URIs) operate at Layer 7 and above in the model, but are not strictly a part of it in of themselves. URIs are simultaneously a machine- and human-friendly method of identifying a resource on the Web, and one may or may not have its own IP or MAC address associated with it. A resource is most commonly a document, but could be other data, such as a video, computer, or application instance, or else a physical object or person.

A URI is particularly useful within the IoT because it can be used to represent a device, component of a device, or a service that uses devices. When queried by a browser, a URI could provide access to the device directly or to a digital twin of the device. Identification of IoT devices is critical to their operation. Unlike in the context of a Web service, which could be replicated across many servers with traffic load-balanced across each of these, IoT devices have to be identified and controlled very specifically.

URIs and URLs

The URI, finalised in 2005 [91], is the generally accepted way of identifying a virtual or physical resource on the Web. A URL (Uniform Resource Locator) is a subtype of URI and, when used by a Web browser, will return an instance of the resource. If returning an instance is not possible, it will return information or metadata about the resource or an error code. In contrast, a URI needs only to identify a resource and may or may not return it. URIs are universal across the Web and are one of the core technologies it relies on.

There were (and in some there cases still are) criticisms of the nature of URIs on the Web. Ted Nelson, a computer scientist and philosopher, is amongst the most vocal, having said that links should be bidirectional [92] meaning that a 404 missing resource error would be impossible. However, the unidirectional nature of URIs is very entrenched in the Web and it would take a significant shift in implementation to change this.

The Semantic Web Interest Group at the W3C puts forward several ways of using URIs with both real objects and their representations [93] and these are being integrated into the W3C's WoT Working Group to represent WoT devices.

2.4 Describing IoT Devices

There have been many alternative ideas for describing devices, from academia, open standards and commercial enterprises. What many have in common is the concept of a 'digital twin' [94], which is a data-based representation of a physical device. Digital twins are especially useful because devices may not always be available. They may be slow to access or offline for extended periods. In these cases, and many others, it can be beneficial to create a skeletal version of the object, one that contains only those features and properties which are useful to the system it is a part of. For example, a digital twin for a light bulb in a smart home may have its power status and brightness as variables as well as the ability to read and change those two variables as functions. In contrast, a digital twin for a light bulb in a machine in a factory may also need to have variables for the number of hours it has been illuminated for, its current temperature, the surrounding humidity, any any other pieces of information relevant to its maintenance and continued operation. This makes it simple to interact with, easy to create a homogeneous interface across manufacturers, and allows for the light bulb to lose connectivity and synchronise with the expected state after reconnection without the user having to resend commands.

The W3C, as keepers of many of the prevailing Web standards, have been developing the concept of a Thing Description (TD) as part of their WoT project. The TD is a JSON-LD [95] document that behaves as a template for a digital twin of a device. It lists several aspects of the device, including its ID, which actions can be performed on it, events it can create, and properties it may have. It is strongly related to the W3C's semantic data projects which allow the linking of entities on the Web to the data that describe them. Using these together, a physical device can be linked to a TD which describes it in terms of the WoT and, in turn, this can be linked to other Semantic Web resources to further describe it.

2.5 The Semantic Web

The Semantic Web [96] is a collection of standards that aims to make data on the Web much easier for a machine to understand. The W3C's proposals and standards allow defining data in terms of one another and connecting them to one another using semantic triples, thereby forming linked data. A semantic triple is not simply a connection, rather it is a connection with meaning attached. A triple could take a form such as 'A belongs to B', or 'X is a daughter of Y', whereby the first and last elements are related data and the middle phrase describes the nature of their interrelationship. These examples are understandable to humans, but with the correct machine-readable definitions, computers can also process them. They provide a uniform data format but, more importantly, allow for machines to infer information by traversing data and relationships. For example, the parent of a parent can be inferred to be the grandparent of the starting node. Using SPARQL [97] complex data can thus be queried and distilled to provide useful information.

Linked data is often stored as RDF [98] or JSON-LD. Both of these formats allow semantic data to be created which references other items within the document, although they also allow references to items from external sources. It is these references that are important for the W3C's WoT Group's implementation and also the future of the work presented in this thesis.

Data vs. Information

For the purposes of this thesis, and following the categorisations often attributed to Russell Ackoff [99], data are defined as raw facts and figures, while information is anything inferred from aggregating or interpreting data. Above these are knowledge, understanding and wisdom. The Semantic Web aims to allow computers to move beyond data to being able to represent information, and then, using SPARQL or equivalent, store knowledge.

2.6 The History of Digital Twins

In some ways, digital twins are an evolution of a specification or technical drawing which, in and of themselves, represent the minimal set of information needed to create or describe an object. However, on top of this static document, they add interactivity via two-way communication between device and its abstraction. They can be used for simulation, as a buffer or cache for commands and state, or as a high-level abstraction of a more complex device. In the 1960s, NASA modelled Apollo 13's [100] systems on the ground when the mission suffered a dramatic system failure. Updates to the live mission were reflected in the equipment on the ground and potential solutions were tested using the various simulators and replicated systems in the laboratories. While this was not formally intended to be a twinning situation, nor was the phrase coined then, it is an early example of one system reflecting another for the purposes of easier access and simulation.

The application to physical products was recognised in a presentation [94] for the manufacturing industry on Product Lifecycle Management by Michael Grieves, and the concept came to be known as the 'Mirrored Spaces Model' soon after. This name is reminiscent of the 1991 book by David Gelernter called Mirror Worlds [101] which described a similar concept. The term 'digital twin' was not used until its introduction in Grieve's 2011 book, 'Virtually Perfect: Driving Innovative and Lean Products through Product Lifecycle Management' [102], having been coined by John Vickers, his collaborator and occasional co-author. The direct application to the IoT grew as the manufacturing sector began to overlap the consumer IoT space.

2.7 Reading State of IoT Devices

The reading of state from IoT devices is linked heavily to both their identification and description. A device's state can be requested by ID or else it can be published by the device via a channel. If a digital twin mechanism is used, then the state can be held within the twin and requested from there. Knowing the current state of a device, keeping it synchronised with its twin, and resolving any conflicts are all key considerations for any IoT system.

2.8 Controlling IoT Devices

IoT devices can be controlled in many ways, although data can only flow in two directions. This means that all control will either occur through data or via commands being pushed to a device, or else through its pulling data from another source and reacting to it. The key difference between these scenarios is the level of autonomy. Where a device receives a command, it will either carry it out, or else perform an assessment of it and decide whether to execute it. However, in a situation where it pulls data, it must decide what to do and whether to do it. This means that actions in the IoT fall somewhere on the spectrum from remote control to completely autonomous devices. There is naturally some division [103] as to whether the IoT is separate to remote control approaches such as SCADA (Supervisory Control And Data Acquisition) [104] as used by manufacturing; whether the IoT is the evolution of these; or if one necessarily encompasses the other. For the purposes of this thesis, a clear distinction is not essential as the primary concern is the underlying modelling of the constituent interactions, rather than the level of autonomy with which they are performed.

2.8.1 Push vs Pull Control Methods on the Web

When the Web was first developed browsers could ‘request’ (or pull) data from a server and the server would return that data via a ‘response’. This remains a very important paradigm today, although it has since been joined by several others. In a pull scenario, the client requests the data they want from a server as and when they want it. However, this can lead to inefficiencies in those instances in which the client wants to monitor data, as it does not know when the data has changed. To combat this, an alternative approach was produced. In modern systems, data can also be pushed to a client from the server using WebSockets [105] or similar technologies. In this push scenario, messages are only sent when the data changes. In some applications, this leads to a significant reduction in the amount of processing being carried out by the client as it no longer has to ask for data that may not have changed in the interim. The server also saves resources as it does not need to deliver identical responses to the same client. In the IoT, this does not necessarily translate to longer battery life or lower resource requirements, but having the option of both methods helps to find lower power solutions.

Polling

Polling [106] is a pull-based approach for a client to get current data from a server. It arises when a system requests the state of another system at intervals, rather than when the user explicitly requests it. The intervals could be regular, or more complex, with exponential or Fibonacci-based gaps depending on the response. In the IoT, a device could poll a URL where it expects to see a command or data to respond to, for example, a weather data feed, or it can poll its own digital twin.

A key advantage of polling is that it can be achieved with very few resources on very constrained platforms, making it an ideal technology for very low power devices that can power on, poll, and then sleep until the next cycle. This is something not readily possible with push systems, as they typically require a constant connection between devices, thereby not allowing the device to sleep.

An early example of a push-based protocol is the SMTP (Simple Mail Transfer Protocol) [107] which was first published in 1982. It is still used to push email messages from one computer to another, usually between servers, until it reaches the users mail box, after which point the end-user will pull it to their device when they connect to their mail server.

A push model was also used in early instant messaging, including IRC [108] which allowed files to be pushed from one user to another. Early data delivery services such as PointCast [109] a dashboard screensaver that displayed various live data, used a similar model. Despite the popularity of push messaging in the 1990s, it was not added to the HTTP specification until HTTP/2 [110] in May 2015.

2.8.2 The Publish/Subscribe Model

Publish/Subscribe, often abbreviated to pub/sub, is a method of messaging wherein messages are usually, but not always, pushed to a list of subscribers to a channel, meaning that only those entities subscribed to the channel can receive them. Messages are either sent to a broker which maintains a list of subscribers or else a list is sent to each subscriber to maintain internally.

A very early implementation of pub/sub was implemented in 1987 by Birman and Joseph [111]. In this paper, the authors use the model for coordinating processes in a distributed system, a phenomenon they called ‘virtual synchrony’. It used what is termed a ‘group identifier’ to represent what we would now call a channel, and this identifier was given to the various group members (or subscribers). Their implementation differs from modern implementations in that it includes the ability to have a shared pool of data. The model itself is similar to a mailing list, which pre-dates this implementation both physically and digitally.

RSS [112] is an example of a combination of pull-based pub/sub, often using polling, which was first created in March 1999 by Dan Libby and Ramanathan V. Guha. A publisher adds messages to an XML document that is analogous to the channel, and a user subscribes to a feed implicitly by obtaining the URL. The user can then poll the document at regular intervals to receive updates, or else unsubscribe implicitly by forgetting the URL. RSS is also a good example of a tiered pub/sub system, wherein some members have read and write permissions and can publish while others can only subscribe and read.

In the IoT, the pub/sub model may create a one-to-many relationship, whereupon one device sends updates to many listeners; or a many-to-many relationship, in which any device on the channel can both send and receive messages to all others. This may be decided informally by the devices on the channel themselves if there happens to only be one producer present, or else formally using permissions for publishing and subscribing. Devices can also subscribe to multiple channels, further complicating the topology.

The concept of pub/sub fits the IoT very well and the use of many concurrent, yet distributed, processes in Birman and Joseph's system maps well to the many distributed devices of an IoT environment. There is no formal definition of how a pub/sub channel should be named or what it should represent, so as a result their properties are completely at the discretion of the owner of the system. In the IoT, channels can be used to represent physical attributes, such as colour or location; or social aspects, such as ownership or purpose.

One of the key advantages of a pub/sub model is that the broadcaster does not need to have knowledge of any of the subscribers. However, this can also be a weakness in that there is no mechanism for receipt confirmations because a broadcaster has no way of knowing how many to expect. In fact, there may not be any subscribers and no error thrown. The only error state in many implementations arises when a broadcaster is unable to send a message after repeated attempts and an undeliverable report is created. Another important benefit of the model is that of its unlimited scale. A channel, as a concept, has no predetermined limit to the number of subscribers it can have, although the system may be limited either by storage of the subscriber list, or by the flow rate for sending messages.

A modern example of a common IoT pub/sub technology is Message Queuing Telemetry Transport (MQTT) [113], an ISO standard. It was first developed in 1999 by Andy Stanford-Clark at IBM and Arlen Nipper at Eurotech and is widely used in IoT applications.

2.9 Historical IoT Implementations

2.9.1 The First IoT Device

The first recorded instance of an IoT device was a Coca-Cola machine [114] in Pittsburgh that was connected to the ARPANET in 1982. It was initially a typical soft drink bottle dispenser and had no inherent Internet connectivity. The only outputs were the lights next to the dispensing buttons which would flash if a drink was successfully dispensed or else turn on continuously if the machine was empty.

On realising that they spent a great deal of time going to the vending machine only to find that the drink they wanted was no longer available (or that the dispensed drink was warm), David Nichols, Mike Kazar, Ivor Durham and John Zsarnay decided to make some improvements. They built a circuit that would poll the lights associated with each variety and recognise when the machine had been restocked. They connected this to the ARPANET and added an interface which showed the stock status as well as whether the drinks were cold (based on a timer triggered when they were restocked). Later, another student connected a nearby M&Ms machine in a similar manner.

2.9.2 Envisioning Connected/Smart Environments

In the late 1980s, Mark Weiser and his team at Xerox PARC were working on imagining a connected environment [115]. This is the first published work that is close to the modern vision of IoT environments, and several concepts are present today. Not all the devices within the team's environment were IoT devices, as some were equivalents of tablet computers and smart whiteboards, but all fed into the same vision. This project also let to the creation of the term 'ubiquitous computing' in 1988. This, and 'pervasive computing', are among the most important precursors to today's IoT.

2.9.3 Ubiquitous and Pervasive Computing

Alongside the Web, other movements were quickly growing in popularity. While the Web dominated the commercial space, in academia both ubiquitous and pervasive computing were gaining support, and the same concept was also seen under the names 'Ambient Intelligence' and 'EveryWear'. The two concepts of ubiquitous and pervasive computing are sometimes differentiated, yet they are often regarded as being synonyms. Both are lenses for the movement that embedded computing devices within environments, which eventually grew into the IoT.

Within this field of research was an area that Weiser and Brown called 'calm technology'. They initially shared it in their 1995 blog post [116], and this concept was later formalised in their 1996 paper [117]. It was derived from Weiser's idea that the data held inside computers would eventually become a part of the environment in the same way that writing is in our visual society. We see writing everywhere on road signs and labels, and we passively absorb the information it contains. We can choose whether to actively engage with it or act upon what it says. Calm technology tried to extend this to live data, and used intelligent environments to disseminate it to the user without them having to directly access or pay attention to a computer. The data could be displayed on everyday objects which the user could choose to focus on, or else could peripherally absorb data from without direct attention. Weiser and Brown's simplest illustration was a piece of string attached to a motor which moved in relation to network traffic. A much better known implementation would be the power and hard disk lights on a computer signalling activation and activity, respectively.

There are some who believe the idea of pervasive computing and, therefore, a great deal of the potential of the IoT has been held back by the creation of smart phones [118] as they tend to centralise control and commercial influences that are often at odds with the notion of frictionless data access.

2.9.4 Coining the Term IoT

The current term, the 'Internet of Things', was chosen by Kevin Ashton in 1999 in a presentation to Proctor & Gamble [119]. However, it was not to be used in mainstream media until a decade later. Ashton claims that this was due to an initial lack of applicability, as data was not stored in 'the cloud', and also because it gained success online because the term 'IoT' was easy to use as a hashtag on Twitter, which saw a large increase in popularity during this period [120].

2.9.5 HP's Cooltown Project

In 2001, a group of researchers at HP published a description of a project they called 'Web Presence for the Real World' [121]. The aim was to tackle the problem of "nomadic" resources or, simply stated, that things and people move around within and between environments.

They thus assigned URLs to people and objects and embedded Web servers into devices that would not normally have IP connectivity. They did this in HP's test environment known as Cooltown, a synthetic space which contained a museum and bookstore. Users were given a PDA that was combined with infrared beacons to track their location. The principle is very similar to modern tracking using Bluetooth beacons which has widespread use in retail.

As the visitor moved through the environment, their experience was augmented by URLs sent to them by the objects they viewed. When the URLs were accessed on the PDA they would show the visitors more information about the object. In recent years, a technically different, yet socially similar construct to this type of augmentation has been implemented in the form of QR codes in museums (Figure 2.8) and on advertising billboards (Figure 2.7). Cooltown is an early example of Web technologies being applied to a domain outside of virtual documents, and one of the first large scale IoT deployments.

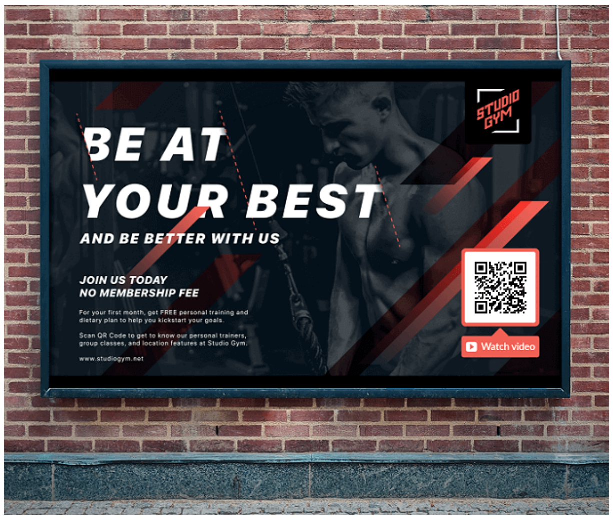

STUDIO GYM
BE AT YOUR BEST
AND BE BETTER WITH US
JOIN US TODAY
NO MEMBERSHIP FEE
For your first month, get FREE personal training and dietary plan to help you kickstart your goals.
Scan QR Code to get to know our personal trainers, group classes, and location features at Studio Gym.
www.studiogym.net
Watch video

FIGURE 2.7: A QR code on a billboard [122].

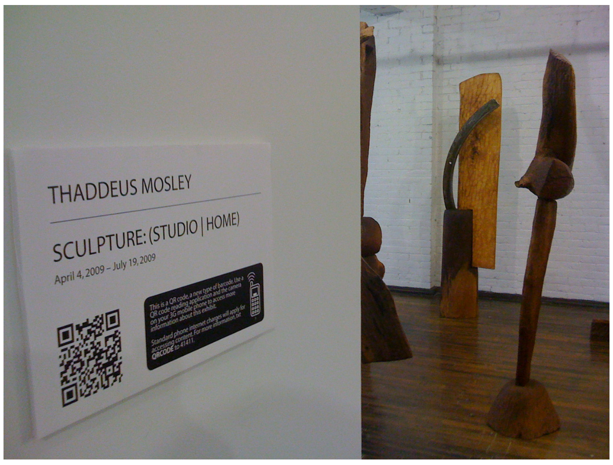


FIGURE 2.8: A QR code in a museum [123].

2.9.6 The First Industrial Physical IoT Products

The Industrial IoT (IIoT) has a slightly different history from that of the consumer IoT, although the two ultimately converge. The IIoT grew out of the Distributed Control Systems (DCS) of the 1970s built by Yokogawa [124] and Honeywell [125]. These allowed the monitoring and control of factory machines from centralised locations and, later, with widespread Internet adoption, also from remote locations.

In the 1980s and 1990s they were integrated with computer networks and, as the Web and consumer interest in the Internet and, subsequently, IoT grew, they became more similar to commercial consumer solutions. The IoT has enabled the construction of increasingly automated factories, as well as saving money within existing factories by allowing the detailed monitoring and management of energy consumption. However, the advent of entirely 'lights out' factories has not yet materialised.

2.10 IoT Management Systems

IoT management systems can be broadly organised using similar classifications to other software projects. Software can be open or closed source and comprise mass market or personal projects. Most open source projects are free, at least at the basic level, while most closed source projects are commercial.

Accordingly, there are four possible pairings, as follows:

2.10.1 Open Source, Mass Market

These are projects created by a group or individual, often with large numbers of contributors later on in their development. An example of this type of software is the Linux kernel, created by Linus Torvalds, which has had many thousands of contributors [126].

2.10.2 Open Source, Personal

These tend to fall into two categories. Firstly, there are those projects designed to solve a specific problem encountered by the author. These are however disregarded from this section due to their highly specific nature, leading to their not becoming full-featured management systems. The second category contains academic projects which tend not to be fully featured, yet contain sufficient features to be classified as a management system when considered with their underlying theories and ideas which complete the picture.

2.10.3 Closed Source, Mass Market

These are mostly walled ecosystems created by a company or a consortium, for example Apple's Home product [127]. They may, or may not be open to extension by developers, and they may be partially open or open at a cost. They may have open source parts, yet have a proprietary piece which is very important to the system. An example of this could be a system with an open source client and a closed source server.

Included in this section are those consortia which produce open source software or standards, but gate keep contributors by charging high fees. They are technically open source, yet lack the communal inclusivity of traditional open source projects.

2.10.4 Closed Source, Personal

These are very rare in the software space as commercial projects generally need to appeal to a large enough market to be viable. These projects are scarce and not otherwise comparable with other systems. An example of this type of project (unrelated to the IoT) is the DeMux project [128]. It solves a personal problem the developer had in that the 2011 MacBook Pro incurred a widespread manufacturing defect in the GPU. The project was then released as a very niche commercial product in April 2019 [129]. However, by their very nature, it is impossible to gauge how many of these projects exist but have never been publicised.

2.10.5 Open Source, Mass Market

Unlike commercial systems, the larger open source IoT systems do not have to worry as much about acquiring and keeping users, and so they are at liberty to find the best user-centric solution to a problem. Commercial systems often adopt a closed ecosystem approach to maintaining a user base, one which is often misaligned with the users' best interests. For instance, it is unlikely that the washing machine and car best suited to the user are made by the same manufacturer, and so the chance of their being within the same ecosystem is reduced. However, open source projects can chart a more inclusive approach. They usually have no outright allegiance to a brand, and even those initially started by a company can be drawn toward a new course by other contributors. The result is that they are freer to cater to multiple ecosystems and devices outside of specific ecosystems. However, without corporate funding, they often do not have full-time staff and instead rely on contributors donating large amounts of their free time. This outcome can impact both quality and support.

Within the IoT space, many of the largest open source projects are orchestration platforms which are designed to bridge proprietary ecosystems using official, documented APIs or, occasionally, unofficial and undocumented APIs. Of these, the biggest are openHAB and HomeAssistant, each catering to a slightly different demographic.

2.10.5.1 OpenHAB

OpenHAB is marketed as "Some Hacking Skills Required" [130] and targeted towards technically-minded people, with the specific aim of customisation over usability. The focus is on the core server implementation and, while there are three user interfaces (UIs) available, the majority of the necessary configuration occurs across many different text files. The control of devices is largely automated, but can also be carried out using the UI.

OpenHAB is designed to connect existing systems, services and applications, including proprietary systems, rather than to replace them. This means that those devices which are accessible to the system will pair and communicate in the way the manufacturer intended. This is important for both stability and usability, but also decreases the amount of maintenance required to keep pace with manufacturers' software and firmware changes.

2.10.5.2 HomeAssistant

HomeAssistant [131] has a greater focus on usability and integration with a single, user-friendly interface. Similarly to openHAB, it integrates with manufacturers' ecosystems rather than attempting to replace them. The key difference is that HomeAssistant is centralised around the UI rather than the server back end. Control is mostly carried out using the UI, with a centralised configuration file for anything that is not present within the visual interface. It uses YAML [132] for configuration, which can be slightly restrictive as it is a descriptive language and not a programming one. It can be used to store configuration data but not to provide scripting or the specific control that openHAB provides. This is instead provided by an automation plugin using a similar 'trigger', 'condition', 'action' schema to IFTTT [133].

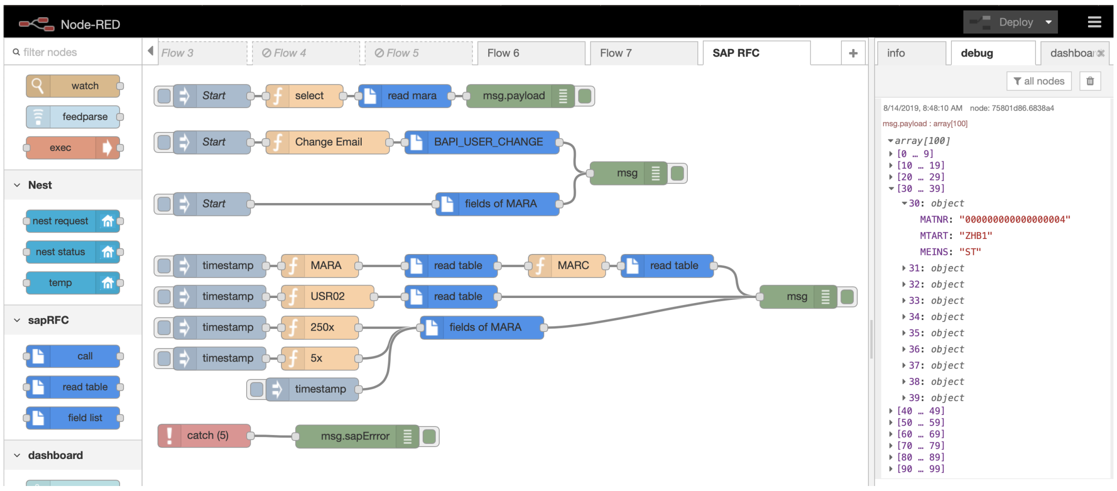
2.10.5.3 Node-RED

Node-RED [134], originally an IBM project but now run by the OpenJS Foundation [135] is a flow-based programming language for automating IoT systems. It integrates with other systems, including openHAB and HomeAssistant, and allows for complex automations based on events and states. It is centred around a visual programming interface which can be seen in Figure 2.9. It aims to be accessible to everyone but also highly configurable and customisable for technical experts. Originally, it was designed to work with MQTT topics, but now connects to any HTTP or MQTT channel and parses data from JSON, XML, YAML and CSV files.

2.10.5.4 WebThings (formerly Mozilla)

As mentioned previously, WebThings was founded by Mozilla but later became a separate entity when funding was removed. The platform is JavaScript-based and uses the W3C WoT Group's implementation of the WoT and adds a layer of usability for developers.

Stark et al. [137] attempted to create a WebThings-based system for controlling devices within a house, with some success. For the moment at least, WebThings is concerned with



```
8/14/2019, 8:48:10 AM   node: 75801d86.6838a4
msg.payload : array[100]
▾array[100]
 ▸[0 … 9]
 ▸[10 … 19]
 ▸[20 … 29]
 ▾[30 … 39]
   ▾30: object
      MATNR: "000000000000000004"
      MTART: "ZHB1"
      MEINS: "ST"
   ▸31: object
   ▸32: object
   ▸33: object
   ▸34: object
   ▸35: object
   ▸36: object
   ▸37: object
   ▸38: object
   ▸39: object
 ▸[40 … 49]
 ▸[50 … 59]
 ▸[60 … 69]
 ▸[70 … 79]
 ▸[80 … 89]
 ▸[90 … 99]
```

FIGURE 2.9: Node-RED's visual programming interface [136].

the identification, connection and control of devices, which is a lower level than the outlined goals of this thesis. However, the approach and ethos of both are aligned in terms of aspirations. Its UI is similar to the second experiment in this thesis (which is discussed in Section 5.2), but the two were developed independently.

2.10.6 Open Source, Personal

Many academic IoT systems fit into this category, but only one uses Web technologies directly applied to IoT devices in a similar way to this project. Amalgam [2] shared many goals and ideologies with this thesis, despite both being conceived and developed in isolation and without knowledge of one another. It aimed to leverage CSS and HTML to describe and build IoT devices at a component level and integrate these within a Web page. It did this by the use of an additional CSS property called `hardware`. This property allowed the developer to specify that the referenced element was a physical device or component. As per this thesis, Amalgam sought to allow Web developers to easily produce code for the IoT although, a Web developer using Amalgam would also need to understand basic electronics to design and program an IoT device.

There are several key differences between the two approaches in that Amalgam aims to aid the development of the IoT device itself, whereas this thesis seeks to interact with pre-existing devices. Further, where Amalgam focuses on the construction of a single device, this project sought to orchestrate the large-scale deployment of devices.

The technical approach of Amalgam is very similar to the approach use in the third experiment in Section 5.3, in that both use Web Components to represent physical objects. Amalgam has Web Components such as `<physical-pot>` which map to existing built-in elements, in this case `<input type="range">`, creating a one-to-one relationship between a UI element and a hardware component. They also recognised the potential for the re-use of libraries as mentioned in Chapter 7.

2.10.7 Closed Source, Mass Market

The closed source, mass market systems include those which are wholly owned by one company, as well as those that are owned and operated by consortia.

2.10.7.1 Nest

Nest, now rebranded as Google Nest [138], began its existence as an IoT thermostat and was one of the first successful consumer IoT products. In 2014, Nest was acquired by Google [139]. Subsequently, in 2020, the brand merged with the rest of the Google Home ecosystem. It has since become a flagship brand for Google's IoT offerings. The underlying technology is largely proprietary, except for the Thread protocol [140] which is jointly owned by the Thread Group. The Nest Thermostat is important because it represents one of the first products to put an emphasis on usability, offering features beyond the capabilities of a normal thermostat. However, it was not originally designed to work as a part of an ecosystem and, as such, could be said to be closer to a remote control over the Web than a true IoT technology. Later, it was integrated with IFTTT which brought it closer to the IoT devices we have today.

2.10.7.2 Philips Hue

Philips Hue [141] is a range of smart lighting that closely followed Nest as one of the first widely successful IoT ecosystems. The first product was released in October 2012 [142] and sold exclusively through the Apple Store. It continues to be a (non-exclusive) partner of Apple Home. It is centred around RGB light bulbs that connect to a network bridge device. The bridge communicates with a Philips server which, in turn, allows the bulbs to be controlled using a smart phone either from inside or outside the environment.

Hue was the first ecosystem to offer scenes, that resemble simplistic style sheets in that they allow the end-user to assign a colour to lights, either individually or in groups, and save that state for later use. Philips also distributed a range of pre-prepared scenes which could be applied generically across any of their lighting products.

2.10.7.3 Google Home

Google Home is the name of an application [143] for controlling IoT deployments in the home. The same name is used for the ecosystem of physical IoT products that can be used with the application. All devices in the ecosystem can be used independently, in collaboration with Google Assistant, and in association with hardware from other vendors. As of December 2020, the brand was merged with Google's 2014 IoT acquisition, Nest [138].

The product is mostly proprietary but offers users and developers the ability to integrate with its system via a public API [144]. This approach is common to nearly every commercial IoT vendor, as it strikes a balance between delivering enough value to the expert customer while keeping commercial intellectual property protected. Importantly, this leaves the architecture and implementation of the system in the hands of the company, which can elect to hinder the creation and development of standards, particularly when a vendor has a strong market presence.

2.10.7.4 Amazon Alexa

Amazon's Alexa [145] assistant service integrates with many IoT devices and services (including their own Echo [146] smart speakers) and provides a management interface centred around voice control. It is very similar to Google Home in its business model, again offering a public API for developers to use. It comes with the benefit of Amazon Web Services (AWS) [147] integration, which offers many more integrations than Google Cloud [148].

2.10.7.5 Apple Home

Apple's Home [127] application is the centrepiece of Apple's HomeKit [149] ecosystem. Homekit is an API and certification programme for third-party IoT hardware, as well as their proprietary HomePod smart speaker. Apple offers voice control through integration with their Siri voice assistant product [150]. They also offer an API [149]. While they lack the connected services of AWS or Google Cloud, they have a very large and loyal userbase from the longstanding and tight integration of Siri with their line of hardware devices.

2.10.7.6 IFTTT

IFTTT (If This Then That) differs from the other large players in this area in that it was originally incorporated in 2010 [151] as a service, and not as a hardware vendor. It links together other online services using simple, reusable scripts, and only later expanded into the IoT when they integrated with IoT APIs from the major vendors aforementioned. It does not produce hardware or IoT products itself, but rather allows users to manage their existing devices within a contiguous system.

As the name implies, the service is a basic programming interface that enables the user to integrate APIs from different IoT services and data providers. It is unique in that it does not aim to provide a directly controlled UI. It does, however, allow products and services, including UIs, from other vendors to be used to trigger routines. For example, Amazon Alexa skills can be created as inputs to IFTTT.

This automation-centred approach sits in stark contrast to Amazon's and Google's products which focus on direct control with an emphasis on voice commands. However, it has commonalities with both Philips Hue and Apple Home which offer their own basic automation steps.

IFTTT is one of only a few services which initially targeted expert users as the end-user of their product and is, by far, the most successful in this domain. However, its enforced simplicity and heavy reliance on integrations built by the IFTTT team means that it is limited in what it can ultimately achieve and is thus far from ideal for developers who may want to attain more complex outcomes.

2.10.7.7 Thread and OpenThread

Thread [140] is a project that was started in 2014 by Google as a way to utilise IPv6 and 6LoWPAN to send messages to IoT devices, but has since expanded into a larger consortium of companies. It is designed to be a mesh communication network that sits between IoT devices to create a network within a building or environment, while having no single point of failure. It links explicitly to the Zigbee Alliance's Matter Project (formerly Project Connected Home Over IP). Its key advantages are that it is standards-based and low power, using IEEE 802.15.4, IPv6 and 6LoWPAN to achieve this.

IPv6 and 6LoWPAN

IPv6 [17] is a replacement for IPv4 [14] which solves the problem of the limited number of IPv4 addresses as well as offering several other enhancements. The IPv4 namespace is limited to 2^{32} addresses, and many of these are reserved for private networks and other special purposes. IPv6 has a namespace that allows addressing of 2^{128} devices, also with reserved ranges.

6LoWPAN [152] is the use of IPv6 over low-power wireless personal area networks, in particular using the IEEE 802.15.4 standard [153]. The aim of 802.15.4 was to produce a wireless standard that could be used by devices with low computational and power resources, and 6LoWPAN allows these devices to use IP. The project was launched with a wide variety of use cases [154] for IoT devices, and this has included deployments from small homes to long-distance sensor networks [155].

Mesh Networks

Mesh networks are an alternative to the star topology, wherein there is no central router that all traffic passes through. Instead, messages are broadcast to all nodes within range and then passed on to nodes within the network which are out of range of the first node. While this can be used in a wired network, this concept has recently gained popularity within the commercial space for wireless networks. The topology allows for easy deployment in a large or noisy space by placing extra nodes until the required signal strength and coverage are achieved. The nodes themselves require little manual configuration within the network, beyond being given credentials to join a secure network, as they are all identical.

This approach is useful for IoT devices, as the node can be embedded within a device and the device placed within an existing space. Provided the node is within range of at least one other node, it will be able to join the network and the process will be almost transparent to the user.

2.10.7.8 Matter

Matter [156], formerly called Project Connected Home over IP (Project CHIP), is a group formed by the Connectivity Standards Alliance (formerly the Zigbee Alliance [157]), with the aim of creating a set of open and free protocols for the IoT. It is backed by Google, Apple and Amazon who, as of 2020, are three of the biggest players in the consumer IoT space. It also has members from several other global IoT vendors.

While Matter is quite new, having only started life in December 2019, it appears to have very similar goals to the W3C, although with a much broader scope. The W3C is concerned with identifying and controlling devices at a semantic level, whereas Matter is attempting to converge entire stacks of multiple vendors from IP connectivity, through security and data models, to the application level. This can be seen in Figure 2.10. As of 2021, they have released some reference implementations and several incomplete specifications, but it would seem that they have a long way to go before achieving their goals.

2.10.7.9 High-End Smart Home Systems

Some systems are targeted at wealthy individuals, and these tend to have more features and integrations than some of the other systems listed previously. However, they come at a cost to usability at the installation and configuration stages. They are designed for coherent whole-home installations and come complete with installation and support packages so that the owner rarely, if ever, has to interact with anything other than the front-end. These

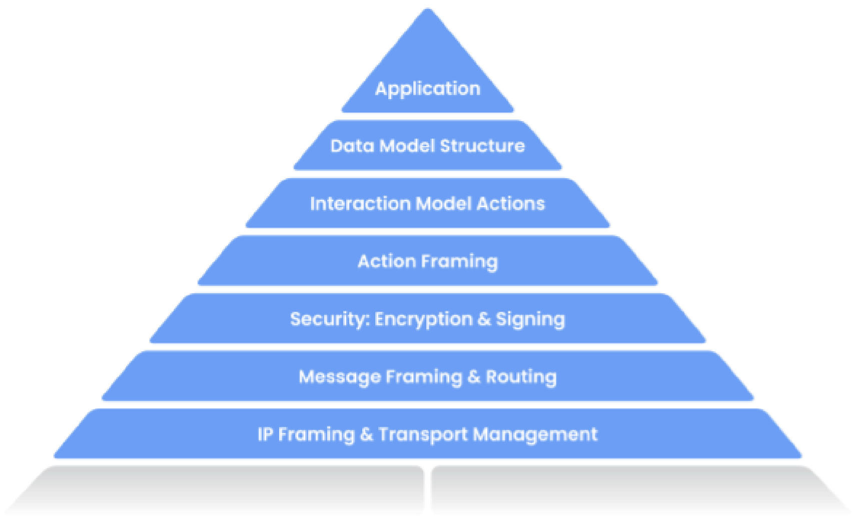

Application
Data Model Structure
Interaction Model Actions
Action Framing
Security: Encryption & Signing
Message Framing & Routing
IP Framing & Transport Management

FIGURE 2.10: The scope of Matter [11].

include Control4 [158], Crestron Home [159] and Savant [160]. These systems are deliberately priced beyond the average consumer and so are unlikely to gain significant market share outside of their niche.

As these systems are installed and configured by professionals, they support scenes much more readily. In the case of Savant and Control4, these are primarily based around presence and situation, offering different setups for parties, relaxation and work. They also offer different profiles for different people, including staff.

2.11 Smart Cities

'Smart cities' is the term used to describe the concept of city-scale IoT deployments of Internet-connected amenities and municipal systems; for instance, a network of smart electricity meters, Internet-connected streetlights, buses, or water and sewerage monitoring systems. The idea is to allow an overview of every system that affects larger groups of people. This should make it possible to reduce inefficiencies, reroute around problems, and coordinate between systems far more effectively. For example, if there is a large event happening in a specific area, then smart rubbish bins could trigger an alert when they need to be emptied; public transport could respond to local demands; and traffic lights could automatically route passing traffic around any busy areas. Smart cities represent the largest IoT environments developed thus far which could potentially include millions of devices. However, in popular culture, the term has been heavily overused wherever a city improves anything related to Internet connectivity.

2.12 W3C and the WoT

The W3C has maintained slow and steady progress in implementing a WoT system since the inception of the WoT Interest Group (WoTIG) on January 20th, 2015 [161].

One of the precursors for this was the webofthings.org Web site [162] which was started in 2007 by EVRYTHNG founders Dominique Guinard and Vlad Trifa. These two people also submitted the original Web Thing Model to the W3C which was published in 2015 [163].

2.12.1 W3C WoT Based Management Systems

As of 2021, the only consumer-facing management systems openly based on the W3C WoT approach are WebThings [86], and Thingweb [164]. The latter is built and maintained by WoT Working Group members. However, judging by the activity of the WoT Working Group [165] there seems to be ongoing development of systems at Siemens, Huawei, Fujitsu, Hitachi, Oracle, Panasonic and Intel.

The March 2021 introduction video [166] shows that the target market for the W3C's WoT project is existing Web developers. It introduces how simple it can be to integrate existing projects with the W3C's WoT implementation, and the promotional video references Thingweb as a flagship example of a WoT library. Thingweb and WebThings both have public-facing documentation which explain the concept of the W3C's WoT approach, including concrete examples of using TDs, actions and events, and reading and updating properties.

Based on the supporter from commercial companies and the standing of the W3C, it can reasonably be assumed that this implementation will eventually become dominant, however it may have to go through an integration phase with Matter for that to be realised.

2.12.2 Matter and the W3C

The Matter project and the W3C WoT project are the two foremost projects at this time. Both have considerable industry and institutional backing as well as sharing many of the same stakeholders and contributors. However, interoperability seems to be almost an afterthought at this stage. The W3C is making the assumption that it will just work with Matter, as Matter is an application layer and the WoT describes application layers using Thing Descriptions and Binding Templates [167].

The Matter project has thus far shown no outward intent to work with the W3C or of developing in line with the other's project. In the initial webinar [168], Jon Harros said that it "may be possible" to collaborate, but that there were no plans to do so. His tone and lack of definite response when asked seemed to suggest that he was unaware of the W3C's project at that time.

2.13 The Human Influence on the Web and IoT

While the Internet and the Web are very technical systems that based on a many aligned protocols that link machines together, their growth and use are very heavily influenced by their human components. Even with the IoT bringing more machine-to-machine communication to the Internet, the vast networks of connected devices are designed to fit within a human environment and, as such, are often designed to make people's lives simpler or more productive. It is also people that provide the majority of programming for the Web. While there are tools, compilers and post-processors that will take a human-readable abstraction and create a more efficient version, the initial need identification, solution design and abstracted code development has been completed by humans.

2.13.1 Metaphors on the Web

Concepts in Computer Science have always been explained to users with metaphors [169]. The reasons for this are twofold.

Firstly, it is known in the field of Philosophy of Computer Science that the originators of concepts tend to create them based on metaphors [170]. Particularly in Computer Science, this often comes about as an explicit desire to replace a physical concept with a virtual one, such as the Bulletin Board System replacing physical bulletin boards. The first example of this was Community Memory [171]; a terminal that allowed visitors to various stores to save, retrieve and print messages. This type of direct replacement naturally leads to the use of the metaphor as it is simpler to replicate the existing methods and terminology, in this case 'posting a message', than to create a new one and retrain the users.

Secondly, both the Philosophy of Computer Science and User Experience Design suggest that user adoption is simpler when the users can relate an idea to one they already know [169] as there is less to learn and so the cognitive load is reduced. This could be a result of survivorship bias as those products and ideas that did not have a relatable metaphor may have not survived.

These metaphors may be very general, such as the notepad, or else be very domain-specific, in the case of the 'save icon' frequently represented by a floppy disk. They may prevail throughout the lifetime of a system, such as the page metaphor used by the Kindle and other e-readers, or else they may be transitional to help introduce users to a foreign concept. Transitional metaphors were heavily relied upon by the iPhone from Apple in the form of skeuomorphic design.

Metaphors may also ‘die’, although this does not necessarily mean that they cease to be used; rather the representation becomes more recognisable than the original source. A particularly famous example is that of younger generations no longer equating the save icon with a floppy disk. The clipboard, address book, carbon copy and folders are other metaphors whereupon the digital concept has replaced the original physical version in most scenarios.

The IoT similarly uses metaphors as per other digital systems. An application for controlling a device may show the user switches and dials that mirror or replace the switches and dials of the physical device it controls, as with Amalgam. In the case of a virtual device, it may show the user a representation of a physical interface, despite the device itself not actually existing.

Skeuomorphic Design

Skeuomorphic design is an approach to design in which the digital representation of a metaphor is visually similar to tat of its physical counterpart. For example, both the Notes and Books applications supplied in the initial versions of Apple’s iOS took advantage of this and were designed to resemble lined paper and a bookshelf respectively. However, this is only a new name for an old concept. WinAmp also used this approach heavily throughout its existence, and the idea of simulated 3D ‘buttons’ in an interface dates back to at least Apple OS 1 in 1984. Examples of skeuomorphic design can be seen in Figure 2.11.

2.13.2 A Cog in a Social Machine

While the Internet and the Web and, as a consequence, the IoT and WoT, can be viewed from purely a technical point of view, they can also be viewed as a part of a larger ‘social machine’ of people and technology. Shadbolt et al. [176] describe these social machines as networks of people and machines that come together to complete a task or support a community. Among the examples given are crowdsourcing maps and those groups that have formed to support those with specific health conditions. The machines may be centred around a single platform, such as a social network, but they extend beyond it, similar to the way Agile Development [177] is adaptive and will use the best tools for the job at hand. In fact, a project using the principles of Agile Development could be framed as a social machine integrating many people, including developers, customers and other stakeholders. It could also include an array of different platforms and technologies to support the development and communication that arises within it. One of the issues with this approach is that it is hard to draw the line between the start and end of a social machine, as it is more of an

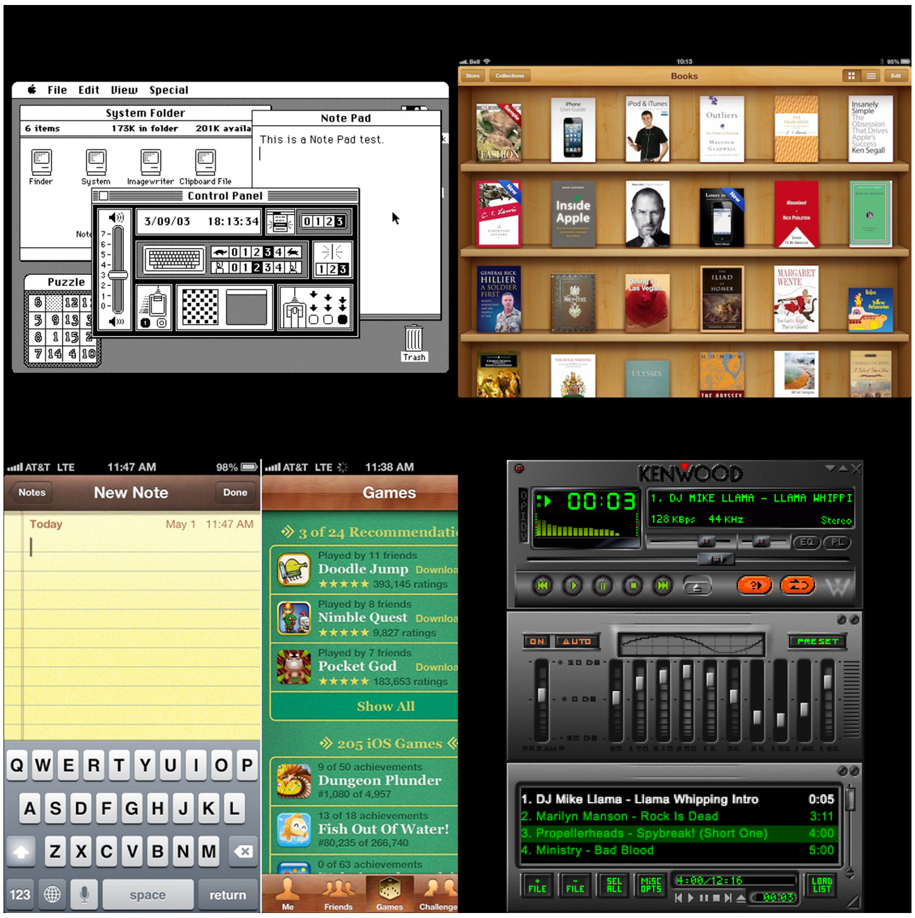


FIGURE 2.11: Examples of skeuomorphic design: Apple System 1 (top-left) [172]; Apple iOS Books (top-right) [173]; a WinAmp skin (bottom-right) [174]; and Apple iOS Notes and Games (bottom-left) [175].

analysis tool than a concrete specification, especially given that many systems involved will not be closed.

Madaan et al. [178] take this a step further into the realm of the IoT and define these systems as Cyber-Physical Social Machines. These machines can be interacted with directly, as with most social machines, and also passively through sensors, to the point that an actor may be unaware that they are integrated within a machine at all.

Using the social machine as a lens is important for this thesis, as the product of it has to be evaluated with respect to the social machine that it could be a part of and not just the technical potential of its construction. While it is essential to know whether the approaches used here are possible to build, it is also very important to know whether they could be used beyond this thesis as a viable part of a larger ecosystem. One of the potential outcomes could be to enable Web developers to transition to new roles as WoT developers more easily, a view shared by the W3C's WoT Group. In which case, any technologies would become a key, though small, cog in a much larger social machine.

2.14 The Future and IoT Environment Design

The IoT and anything surrounding it will need to consider both the technical problems that come with scale as well as the human problem of explaining new paradigms in a relatable way. It will also have to cater to the audience, using metaphors that are broadly understandable to the general user, as well as metaphors that are useful in specific domains. Even in highly technical domains such as programming, metaphors will help people to understand and implement key ideas in the IoT.

Despite the scale of the IoT increasing every year, deployment sizes are likely to form a power law, as for many other aspects of the Web. This means that while large-scale deployments have to be catered for, many – if not most – deployments will be smaller in scale. Any potential solution has to provide for all of these potential deployment equally well.

2.14.1 The Case for Open Standards

Open standards enable users to take control of their devices, a notion that can be contrary to the interests of the businesses that produce them. Apple, for example, has long been a proponent of a 'closed ecosystem' wherein users can only use their devices in conjunction with other Apple devices and services. Before he passed away, Steve Jobs emphasised this in a 2011 internal memo [179].

An important issue within the IoT is support for older devices and this can be aided, although not solved, by open standards. There is little reason for a company to keep supporting a product that is no longer profitable or else offers non-tangible benefits to them, thus leading to obsolescence. Among the outcomes of this lack of support are the manufacturer deciding not to upgrade firmware for continued compatibility and security; not providing parts to repair it; disabling a cloud service it relies on; or simply going out of business. Open standards allow the expert user to mitigate some of these scenarios and potentially help novice users in that, at the very least, they can replace missing cloud services with alternatives or else produce generic tools.

Obsolescence is a serious issue for typical computing-based devices such as mobile phones and smart watches, but is arguably even more complex for IoT devices. It is perfectly reasonable to assume, where a device is composed of a mechanical device and a computer, that the mechanical part could outlive the computer or the cloud service the computer links to. However, if the device relies on the computer or service to function at all, then this leads to inconvenience for the user when the service or computer fails. There are plenty of examples of kitchen devices and tools which have survived since the 1950s on Reddit's 'Buy It For Life' subreddit [180]. While there is obviously a large amount of survivorship bias [181] involved, this does not negate the fact that devices have survived for 70 years and continue to function, while IoT devices today would have likely suffered either a failure of their internal computers or those services upon which they rely. Additionally, there is an environmental factor to this, as an average user would have to dispose of a device that is no longer supported, despite the mechanical parts being in a potentially functional state.

The Little Printer

One early example of IoT obsolescence was the Little Printer [182], a receipt printer repurposed as an ambient display and news feed. The company shipped the first items in 2012, yet the product did not survive beyond 2014, leaving a lot of users with an unusable, but still perfectly functional device. A few years later in 2019, a digital studio brought the Little Printer back to life as a promotion for their business [183]. However, this was only possible due to the work of Matt Webb in hacking the device and producing the open source Sirius Server [184] for it, which he was only able to do because the device itself used open standards for communication. This type of approach is a workaround to the problem and not a solution per se, but becomes impossible with devices that have been actively protected from tampering using encryption or even simply hard-coded URLs. Some devices go further and employ closed or modified versions of standards in their hardware, such as many of Apple's chips used for communication in their headphones and watches.

AWS Outage

While the loss of services may not be permanent, even a temporary service outage can cause a lot of problems. In November 2020, AWS (a major supplier to thousands of IoT businesses) had a major service outage [185], causing many IoT devices to stop functioning for the duration of the issues. Devices which were more open could have readily switched to an alternate service, but such devices are at odds with the ethos of systemic control as practiced by many of the large IoT players.

2.14.2 The Case for Re-Use of Web Standards

There is a recurring theme in the literature that the re-use of Web standards should form the core of the WoT [186]. However, the direction of the industry – as shown in the previous section – indicates a disparity with this. Companies have again chosen to produce closed systems akin to AOL's 'walled garden' of the late 1990s [187] which was widely regarded as a bad move.

They have thus each created their own closed ecosystem with little or no active development supporting interoperability, and this has already led to a fragmented marketplace in which it is nearly impossible for two devices from two different manufacturers to communicate seamlessly. Matter may help or even solve this, but it does not yet have the momentum necessary to overwhelm the competition, and its lack of cooperation with open standards groups, together with the large financial barrier to entry, will likely hinder universal acceptance.

Re-use of Web technologies could help in this area, as the initial push of open standards overcame the walled gardens of the past. Specific to this thesis is the notion of re-use of the DOM and HTML, JavaScript, and CSS, as they represent the core of the Web. People have long attempted to envision the use of CSS in other situations, and even Håkon Wium Lie's PhD asked whether "style sheets describe presentation in domains other than electronic documents?" Håkon Wium Lie's PhD [7]. One example he gave can be seen in Figure 2.12.

```
Norway Oslo Drammensvn 97 b {
  floors: 3;
  color: #FCA;
  roof: mansard;
}
```

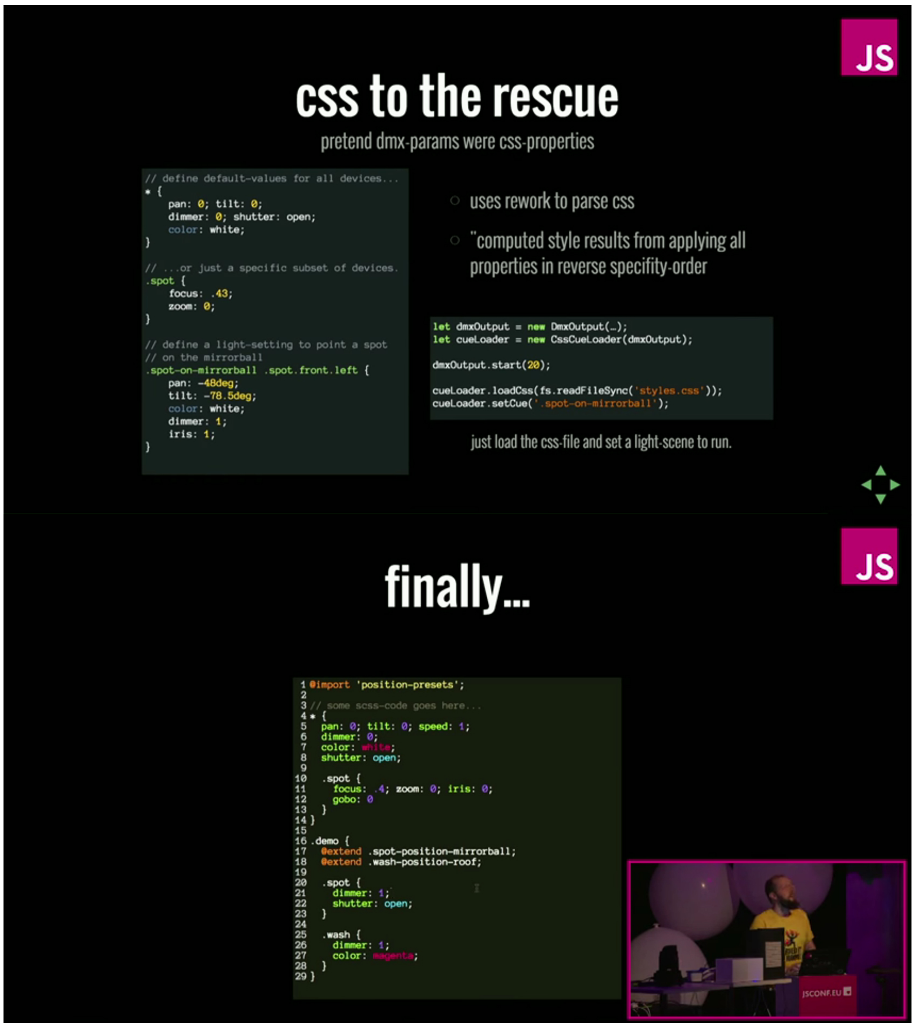
FIGURE 2.12: An example of CSS being used in a non-document domain, from Håkon Wium Lie's PhD Future Work chapter [7].

The CSS specification itself allows for the typesetting of physical documents, which is the only supported use of the standard outside of digital media. It offers several features, including page numbering, physical units of measurement, print layouts and odd/even page selectors, although many of these lack proper implementation in most of the systems that use them. One of the very few systems to use CSS for typesetting is the proprietary Prince: Print with CSS [8] product, whose chairman is Håkon Wium Lie.

The specification also allows for styling spoken output using the Speech Module [188]; including defining pauses, tone of voice, volume, pitch, gender and voice family. The latter is the equivalent of a font for a voice. While this is largely an accessibility feature, it shows support from the maintainers of CSS for the standard to be used outside of simple typesetting.

Other examples of using CSS outside of digital media are scarce, but in 2015 Martin Schuhfuß gave a demonstration entitled 'Let there be light' [1] at the commercial JSConf where he used CSS to overlay the DMX protocol [189], as seen in Figure 2.13. As part of the demonstration he controlled the stage lights in real-time using CSS commands.

Another case of thought which runs parallel with some of the concepts introduced in this thesis is an article by Mate Marschalko [190] from 2017 in which he presents the idea of



css to the rescue

pretend dmx-params were css-properties

```
// define default-values for all devices...
* {
    pan: 0; tilt: 0;
    dimmer: 0; shutter: open;
    color: white;
}

// ...or just a specific subset of devices.
.spot {
    focus: .43;
    zoom: 0;
}

// define a light-setting to point a spot
// on the mirrorball
.spot-on-mirrorball .spot.front.left {
    pan: -48deg;
    tilt: -78.5deg;
    color: white;
    dimmer: 1;
    iris: 1;
}
```

- uses rework to parse css
- "computed style results from applying all properties in reverse specifity-order

```
let dmxOutput = new DmxOutput(…);
let cueLoader = new CssCueLoader(dmxOutput);

dmxOutput.start(20);

cueLoader.loadCss(fs.readFileSync('styles.css'));
cueLoader.setCue('.spot-on-mirrorball');
```

just load the css-file and set a light-scene to run.

finally...

```
@import 'position-presets';

// some scss-code goes here...
* {
  pan: 0; tilt: 0; speed: 1;
  dimmer: 0;
  color: white;
  shutter: open;

  .spot {
    focus: .4; zoom: 0; iris: 0;
    gobo: 0
  }
}

.demo {
  @extend .spot-position-mirrorball;
  @extend .wash-position-roof;

  .spot {
    dimmer: 1;
    shutter: open;
  }

  .wash {
    dimmer: 1;
    color: magenta;
  }
}
```

FIGURE 2.13: Captures from the video of the 'Let there be light' presentation demonstrating the utility of CSS and SCSS to control stage lights using the DMX512 protocol [1].

using XML to represent environments and CSS to control them. He calls these IoT Markup Language (IOTML) and IoT Style Sheets (IOTSS), respectively, and the concept is very similar to some presented within this thesis, yet is more limited in scope. His approach uses a hierarchy of DOM elements in XML to represent devices and attributes attached to the elements to represent their state. In many ways it is similar to the first iteration of the first experiment of this thesis which was presented [191] at a commercial SkillsMatter conference in October 2015, initially called Real CSS.

Re-use of these specifications could not only prove faster than producing new ones, but also allows the re-use of existing tools and frameworks, thereby requiring less effort from existing Web developers to make the transition between domains. It also goes some way to treating the WoT as a social machine, one which is evolving from the current social machine of the Web rather than just being a mere technological advancement.

2.15 Limitations of Existing Research

The prevailing systems seem to be converging on an outcome wherein the W3C's WoT system will be able to describe many other systems and, most likely, Matter will be the dominant consumer system with a fairly comprehensive industry backing. Alongside this will be several bespoke systems, and the high-end systems will likely continue to operate as proprietary IoT implementations. However, there are a few challenges that these current and near-future projects cannot overcome easily.

The commercial projects are very focused on solving those issues faced by the backers of the specific project while keeping control within the project. Matter is a good example of this, as a certain amount of it is open source, while a lot of its standards and implementations are closed [11]. However, membership of the group is prohibitively expensive and it is the group members that ultimately define and drive the goals. Their unwillingness to work with other groups – like the W3C – is fairly typical of commercial ventures. Consortia such as this have the joint aims of creating marketable goods, while simultaneously creating barriers to entry into the marketplace through secrecy and fees.

Academic research in the area investigates the very granular technical problems or else studies large social issues brought about by existing developments. This is very useful, and the findings are often employed by larger groups, although this approach rarely looks at the larger picture of interoperability, and almost never considers the end-user or implementer.

Other organisations, like the W3C, focus more on the implementer and aim to create a more homogeneous system, one that follows expected best practice. This is an approach that favours the implementer with the hope that it will create a predictable and consistent experience for the end-user. However, in this case, they have chosen to discard a great number of their own potentially useful technologies in favour of a much more semantic approach. While this seems to be a very sensible and well thought out approach, it risks alienating existing Web developers, as their skill-set will not easily transfer to this new set of standards. Many Web developers have no formal Computer Science training, and some of the concepts involved in the W3C's approach require an academic knowledge of entity relationships and semantic data, amongst other things. This is a mindset that may not be compatible with more visual-thinking and design-oriented Web developers.

In the following chapters, this thesis attempts to avoid some of the highlighted pitfalls of the ecosystems and methodologies outlined in this chapter, through an alternative to the W3C's mental model for the WoT. Re-use of the existing standards of HTML, CSS and JavaScript, that have so far been ignored by the W3C's WoT Group, could allow for control of environments containing IoT devices using the metaphor of a Web page. The hope is that this could be a palatable transition for Web developers, while being compatible with the Web as it exists today. There is no implication that it should override or replace either commercial implementations or the W3C’s approach, but it could potentially replace aspects of them, or simply provide a familiar overlay for a complex new technology.

Chapter 3

Theoretical Framework

This chapter explores the high-level challenges of the thesis; the knowledge gap it addresses; how it addresses both; and why it is important to do so. The aims described in Section 3.3 are threaded throughout this thesis from methodology and planning, to experimentation, and finally to the discussion of the potential impact of the resultant findings.

There were three aims identified as key stages in the development of the approach taken by this project. Collectively, these aims target the theoretical, practical and social challenges of mapping IoT environments to the DOM. They were executed through four key experiments (discussed in detail in Chapter 5) which encompassed: a theoretical approach in Experiment 1; a working implementation in a real environment in Experiment 2; a more refined simulation of multiple environments in Experiment 3; and finally, an assessment of the idea through the eyes of the wider development community in Experiment 4.

This chapter sets the scene for the following chapters, and is followed by Chapter 4, containing decisions that were made before experimentation could commence, and Chapter 5, which describes the approach, results and conclusions of each experiment in detail.

3.1 Challenges

The key opportunity mentioned in the previous chapter is an approach that allows Web developers to use their existing knowledge to program for the IoT. This gap exists because the systems described thus far all struggle to overcome a couple of challenges.

All of the offerings are, to some degree, external to the Web of documents. For example, many IoT devices are accessible through the API of a server, hub or device. This means that they may be controlled or queried using a request from a browser, and data can be shown in the browser, yet they are not intrinsically a part of the document in any way. The W3C WoT

Group's system allows a description of a device to be requested in the form of a TD, then the device can be controlled or events subscribed to, although the device remains external to the extant Web. This leaves a gap for a system that more tightly integrates the Web and IoT, for example by allowing a device to be placed on an even footing with a block of text or a video within a Web page. In a similar manner to how a `<video>` element is placed in a Web page brings an external video resource into the DOM, it would also be possible for an element representing an IoT device to bring an external physical resource into the DOM. The re-use of technology in this way could save developers both time and effort.

Existing IoT solutions require a substantial body of new knowledge to be learned, which is a barrier to entry that incoming IoT developers must overcome. Even in the case of the W3C WoT Group's project, which aims to be Web-native, a different mental model has to be adopted in relation to other Web development. Each approach and system requires Web developers to significantly increase their mental burden, at a juncture when they are already strained by a large number of ever-evolving frameworks. The consequence of this could be a wholesale rejection of the technology or else the forcing of developers to specialise into a narrower niche. In doing so they would either suffer increased stress, or else limit their opportunities considerably. An alternative solution could narrow the gap between the mental model for a Web page and that of an IoT deployment, and approach which could hopefully mitigate some of these negative outcomes.

While neither of these challenges is likely to be an insurmountable barrier for an implementation or the development of the WoT as a whole, it is possible to do better. Through the re-use of existing technologies and standards, it will hopefully be possible to create an approach that could simply repurpose existing developers toward new WoT projects using their current skillsets. We treat IoT devices as a part of the Internet and so, by extension, we may also be able to treat the WoT as an integral part of the Web.

Guinard, Trifa and Wilde [192] made a first step towards this by describing a WoT system which connected devices using REST [193] rather than RPC [194] approaches. However, there exists a chance now to go a step further and bring the devices into Web documents. Guinard, Ion and Mayer [195] recorded the phrase from a developer that "Everybody who is using a browser already knows a little about [REST]", and it follows that everybody who is using a browser also already knows a little about the DOM.

3.2 The Gap

As already alluded to, there exists a gap in the WoT wherein devices become more tightly integrated within the Web and particularly within Web browsers. Specifically, representations (or digital twins) of the devices can be slotted into a DOM as nodes in the tree. These twins are then mapped to real world devices, and thus any change to either is reflected in the other. The DOM in question may or may not also be a Web page, but if it is, then the device elements could be treated transparently and with equal weighting to any other element.

The use of the DOM also opens up the potential to apply many other Web technologies, but this thesis explores only HTML, CSS and JavaScript. It focuses on the use of CSS to represent and control the state of an IoT environment in much the same way as it is used on a Web page to control the visual state.

The potential for this approach is very obvious for presentational and visual properties of an environment, such as the colour of lights, which was briefly explored by Martin Schuhfuß [1], but does not have to be limited to this. As Håkon Wium Lie alluded to in his thesis [7], CSS can be applied to much more than just Web documents. There is no technical difference between controlling the colour of a light, the temperature of a refrigerator, or the height of a block of a stage. As yet this approach is an almost completely unexplored area, yet one with huge potential. This thesis demonstrates how an environment represented using a DOM can be controlled and monitored using CSS and JavaScript in ways that have never been attempted before. In doing so, a truly Web-native approach is explored, implemented and tested.

3.2.1 Specifics and Justification

Simply having a gap in knowledge is insufficient to justify a thorough investigation, as there must also be a technical or social reason to explore that niche. In this case, there are both technical and social reasons for doing so.

3.2.1.1 Technical

Technically, current approaches to controlling the IoT are inefficient and relatively naïve. They are inefficient in that, for the most part, they do not take advantage of many of the benefits that existing Web technology has to offer, and naïve because they offer very limited control over the devices within their respective ecosystems. Commercial systems are particularly guilty of these failings.

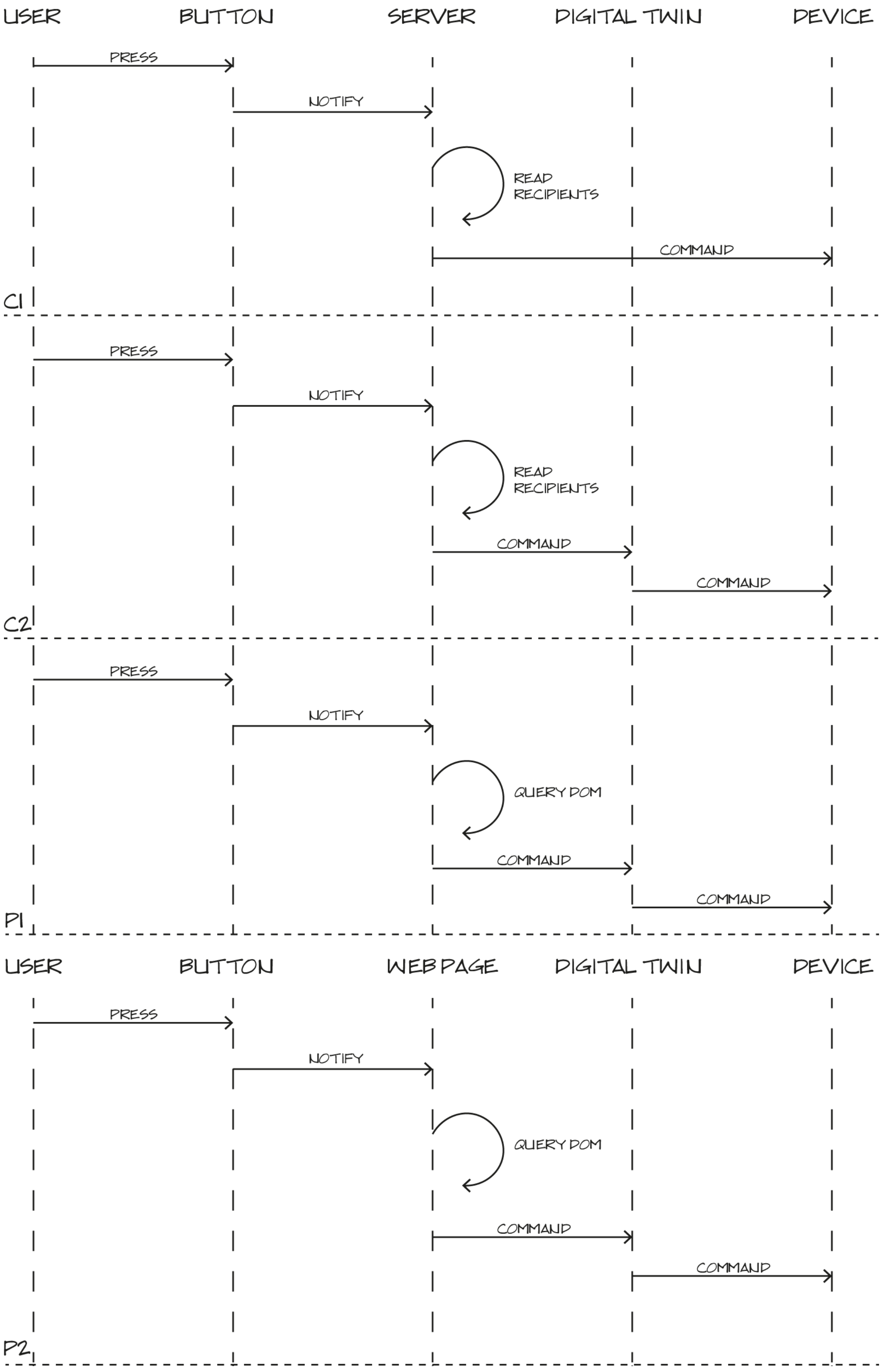


FIGURE 3.1: Situations Current 1 (C1), Current 2 (C2), Proposed 1 (P1) and Proposed 2 (P2), in which a user presses a button which then acts upon a device.

Considering the most basic case of a device being turned on by a button, modern IoT systems use two messaging approaches for remote control (C1 and C2 in Figure 3.1) which both rely on a hub or server. A third method also exists, wherein the user's application sends a message directly to the device without an intermediary. This type of direct control is less common commercially as many users want remote access and it is simply more secure to do this via a hub or hosted service. With the advent of IPv6 [17] and 6LoWPAN [152] we may yet see more direct control without hubs in the near future, once the security issues of doing so have been overcome.

Figure 3.1 shows the two current server-based situations for messaging an IoT device using the Web (C1 and C2) and, also two other proposed alternatives (P1 and P2). These proposals attempt to overcome the failings of C1 and C2 by taking advantage of existing Web technologies. Both C1 and C2 use Web and Internet technologies, but for the most part, they fail to take advantage of the scalability, granular control, or interoperability that the Web offers. Moreover, they also ignore the vast amount of optimisation and pre-existing code that has been written for the Web and Web browsers.

Both C1 and C2 require proprietary code to be written for each API route, which from a development point of view can be complex, repetitive and time-consuming. A developer would need to produce a hub API with many routes, a UI that calls these routes, and devices with their own APIs called by the hub. P1 improves upon this by using the DOM as the engine for all routes, meaning that the developer would only have to write the interface between the hub API and the selection engine once, and this would work across many scenarios. P2 moves this selection away from the server and into the domain of the browser which already contains a DOM, so the developer no longer needs to have a central hub (or else only a very basic hub) and instead must just write the device APIs.

C1 and C2 demonstrate a loss of control for the user whereupon their action is not linked directly to the outcome. The action the user takes could be the same, but the server might perform any action when a specific route is called. The control and power rest entirely with the hub software and not with the user. P1 does not improve this scenario, but P2 places the power in the user's hands. While this is unlikely to affect novice users, it could be very useful for advanced users and developers. Both P1 and P2 offer granular control over devices within the DOM, although P1 requires the developer to trust the hub's implementation of the DOM.

C1 and C2 require that the button makes either a single call to one server, or else need custom code to be written to contact multiple servers with multiple requests. P2 allows for all communication from the UI to be delegated to the digital twins which should already be linked to their physical counterparts. In this situation, the button code does not need to be customised beyond a selector which chooses which twins to send the command to.

P2, once fully realised, would enable control of the IoT with almost no proprietary code. This would be far quicker more robust. Use of the DOM enables massive scalability and granular control, which can be aided by the use of CSS selectors to direct messages to sets of devices. Interoperability comes with this by default, not only between devices within the same DOM, but also between existing documents and code libraries.

P1 and P2 are explored further in Experiments 1-4 in Chapter 5.

3.2.1.2 Social

Current systems typically require a developer to master a new system or paradigm or, at the very least, a new API. While developers are very adept at doing this, each new approach forces them to either generalise and increase their mental burden, or else to specialise and reduce their employability. Within the front-end development space there are currently three competing frameworks, namely React [37], Angular [36] and Vue [38], with many smaller frameworks of varying popularity. Each of these follows a similar approach, yet they are all sufficiently divergent that experience with one does not necessarily mean that a developer will be able to work with another without acquiring new skills. However, as they are based on the same underlying technologies, it is possible to move relatively quickly from one framework to another. The IoT space has yet to reach this level of complexity, as most interactions are from one user to one device and employ relatively simple APIs. Although, as the complexity of the environments increases, it is not unreasonable to assume that the complexity of control frameworks will also increase. Treating the WoT as a new problem requiring new solutions only increases the number of skills a Web developer must learn.

Socially, this presents something of a problem. Provided there are a finite number of developers and assuming that they would generally rather focus on a niche area than suffer the stress of trying to know everything, then forcing increasing specialisation upon them will lead to a fragmented pool of talent with fewer developers able to perform work in each area. The re-use of extant Web technologies could aid this approach by not forcing a division to arise between the Web and the WoT. Directly applying such approaches and tools of the Web to the WoT could make it possible for developers to become both Web developers and WoT developers. The analogy is somewhat similar to React and React Native [196], which share a common approach and, for the most part, a common language, but which are applied to quite different problems.

This strategy could also have a cascading benefit to less technically proficient users, as it may be more likely that generic and user-friendly tools and applications could be built if more developers are focused on a smaller number of technologies. In this case, the market competition would be shifted from competing standards to competing implementations; an echo of the change which benefited the users in the first browser wars [197]. Once all the browser implementers began using the same standards, they began to compete in terms of usability and supporting features.

3.2.2 How This Fits Into the Timeline of the WoT and IoT

While this initially seems to be a parallel development, at least when compared to the current state of the IoT, it can also be seen as filling in a few missing niches and allowing for a natural progression from naïve young technologies into more mature and fully featured ones. It seems most likely that it could do this by becoming a standards-compliant layer on top of the W3C WoT Group's implementation which is, in itself, a layer on top of other IoT control systems. However, this level of integration is beyond the scope of this thesis, one which instead aims to validate whether using the DOM and CSS is a sound approach and therefore worth pursuing.

The literature review has revealed that there has been a logical progression to realising the Web we use today, and this can be seen mirrored in the progression of the IoT and WoT. Both follow a few linear steps and both share important stages in their development; pioneered as proprietary solutions only to be subsequently replaced with a standards-compliant and open version. The two timelines are compared in a very simplified and tailored way as outlined in Table 3.1.

The Internet and the Web	**The IoT and WoT**
The Internet began life as two mainframes connected over a large distance. This proprietary protocol evolved into the Internet Protocol, an open standard.	The IoT began its journey as a computer and drinks machine connected over a (much shorter, but still significant) distance. This proprietary approach of probing an open port has been replaced with REST APIs and sockets.
The Internet grew to many connected mainframes which communicated with each other.	The IoT grew from single device control, to multiple devices communicating with each other within an environment at Xerox PARC [115]. Later deployments have used proprietary implementations, although more recently they are being replaced with Thread, Apple Homekit and other commercial (but largely open) standards. Soon they will be replaced with Matter, which is another step towards a universal, yet still commercial, standard.
The Bulletin Board System allowed for the direct connection to stores of documents through a dial-up interface.	IoT ecosystems allow for direct connection to hubs of devices through sockets and APIs.
The Web uses the DOM to represent structured groups of document components.	IoT systems have many ways to group devices, including lists of channel subscribers in MQTT, although there is nothing as structured as the DOM.
The Web creates a network of documents that can be navigated through links. In the early days there were many discussions about how links should be implemented, which led to a few competing options put forward by various influential people. To this day there remains disagreements between browser implementers and the W3C about the direction of new features.	The W3C's WoT project creates a network of devices and TDs that can be navigated through links. This could replace a number of proprietary digital twin-based systems.

The Internet and the Web	**The IoT and WoT**
The creation of CSS allowed for styling and theming Web pages, but this emerged from a set of other competing and precursor technologies.	This represents a key divergence as there is not yet an open or universal way to style or theme the IoT. Many systems offer the scene concept which allows for styling and theming of IoT environments in various proprietary ways, yet are quite restricted in their features.
The introduction of JavaScript in the browser allowed for dynamic interactions with Web page elements and data sources.	Projects such as Node-RED and IFTTT allow for scripting of interactions (and automation) with the IoT, but these will likely be replaced or augmented with the W3C's WoT Scripting API.
Many systems and frameworks have been created and layered over the Web since the mid-1990s to produce large interactive ecosystems. Doing so would not have been as straightforward without the open and free nature of the protocols and homogeneity of the process of development for the Web.	The IoT and WoT are far from this level of developer engagement as there is not yet that level of homogeneity. There are several systems with large amounts of developer engagement, for example Amazon's Alexa platform. This, however, is closer to AOL's walled garden of information than it is to an open equivalent like Wikipedia.
These layers over the Web have enabled the Web 2.0 push, as well as the potential for non-technical content creators to engage directly with the Web and their audiences.	The IoT and WoT are also quite a long way from this, with current scene sharing being distant from even a MySpace or WordPress theme of the early millennium.

TABLE 3.1: A comparison of key events in the Internet and Web with the IoT and WoT.

The gaps in the current WoT vision, as highlighted in Table 3.1, are a lack of a DOM equivalent and the dearth of a CSS equivalent. As a result of building these, it is conceivable that a barrier could be lowered for developers wanting to engage with the IoT.

Representing IoT devices in a DOM could open up several avenues for more closely mirroring the progression of the Internet and Web and aligning them with other technologies in that chain. Use of the DOM may require a few tweaks, but once complete the device and its components would be in the same format as a section of a Web page and its children. This opens the WoT to the use of CSS, or else a close cousin, in order to provide a more standardised way of providing themes as well as control over larger deployments. These, in conjunction with JavaScript and potentially the W3C's WoT Scripting API, could mean that not only is there a WoT 'piggybacking' on the Web we already have, but also that existing frameworks and tools may only require minimal changes to allow them to work with the WoT as well. Hopefully, a by-product of this thought process might yield a vastly simplified mental model of development for the WoT. This could possibly even save many years of unnecessary re-development in building new frameworks for a new WoT by instead re-using existing technology through a compatibility layer sandwiched between the IoT and the Web.

3.3 Aims of This Thesis

The fundamental conclusion from the literature review is that there is a common arc to the development of the Internet and Web. It is the underlying assumption of this thesis that the IoT and the WoT share a similar trajectory. Further, there are some gaps in this arc for the WoT which could be filled using technologies from the Web. With this in mind, this thesis will attempt to push the progression of the WoT closer to the progression of the Web via three overarching aims:

Aim 1: To treat IoT devices as we do the elements of a Web document by representing them within a DOM

Aim 2: To build a system for controlling and monitoring IoT environments using only browser technology, ideally with CSS and JavaScript

Aim 3: To produce an approach which is acceptable to existing Web developers that could allow them to easily transition into WoT development, thereby following existing best practice for Web development

These three aims are addressed further below:

3.3.1 Aim 1: Treating IoT Devices as DOM Elements

The building blocks of a Web page are the HTML elements of which it is comprised. Similarly, the building blocks of an IoT environment are those devices of which it is composed along with their components. Treating an IoT environment of devices similar to a document of HTML elements allows for an application of existing Web technologies. Notably, devices could be added to a DOM in a hierarchy, and the resulting tree would be a representation of the environment in which they exist. This was explored initially in Experiment 1 and continued throughout the other experiments.

Experiment 1 was a first attempt at modelling an IoT environment using the DOM and took advantage of XML elements to represent an environment's devices and the components of those devices. It did so as a hierarchy which was structurally identical to any other XML resource, and similar to a Web page. While the XML approach was replaced with HTML in the real environment of Experiment 2 and the simulated environments of Experiment 3, the devices and their components continued to be represented using DOM elements. Experiment 4 used the simulated environments of Experiment 3 and therefore also aligned with this aim.

3.3.2 Aim 2: Building an IoT System Using Browser Technology

Following on from the previous aim, once the IoT devices are represented within a DOM, it is possible to apply the existing technology in browsers to both monitor and control them. This would afford considerable potential for the significant re-use of existing approaches in keeping with the prevailing mental model that Web developers use to build the Web today. Key applicable technologies are HTML, CSS and JavaScript as they are ubiquitous in Web browsers and were originally present as key ingredients of the Web at an equivalent developmental stage as the WoT as it stands today.

Combined, the DOM, CSS and JavaScript complete the arc identified earlier and bring the WoT more into line with the Web, an outcome that could bring significant benefits. This was first tested in Experiment 2 and was taken further by Experiment 3 and 4.

Experiment 2 was the first to implement a working system, and did so using a headless browser to house and interact with the digital twin of the environment and the twins of devices within it. This initial version was limited by the available technology, but was vastly improved upon in Experiment 3, which was almost entirely browser-native. Both used a combination of HTML, CSS and JavaScript, and used each of the browsers' native APIs for managing the twins they contained.

3.3.3 Aim 3: Gaining Acceptance from the Community

Many of these benefits could only be realised with widespread acceptance. They may include the creation and adaptation of frameworks, tools and applications that currently use Web technologies; all of which would require significant investments of time and money. It is assumed that acceptance is heavily impacted by how easy it is to transition to a new process, as a small change will always take less effort than a large shift.

Acceptance could also ameliorate the social issues of current approaches. If an acceptable solution can be found and adopted, then this could become a unifier and reduce the level of fragmentation within the development space.

This aim was principally investigated in Experiment 4 where members of the development community were asked to explore the WoT system developed in Experiment 3 and reflect upon the suitability of the approach. The results were a combination of observations of their performance and the shift in their mindset during the experiment, followed by their subjective feedback regarding the experience.

3.4 Scope and Assumptions

This thesis is concerned only with the level of technical complexity that a Web developer would regularly encounter. This includes code written using HTML, CSS and JavaScript. For the most part, the inner workings of the browser, operating systems and device firmware are not considered, and neither are proprietary layers or languages. Also omitted from consideration are those technologies such as Web Assembly [77] which appeared much later in the lifecycle of the Web. With respect to the OSI Model [88], this would be the application layer (Layer 7), with some consideration of the presentation layer (Layer 6). Beyond this model it also addresses the needs and opinions of the potential expert users.

The Open Systems Interconnection (OSI) Model

The precursor to the model was originally conceived in the 1970s, but took shape as the OSI model in 1984 when it was jointly published as ISO:7498 [198] by the ISO, and as X.200 by the International Telecommunication Union [199]. Both the original documents have since been withdrawn and replaced by updated versions. While not a perfect model of modern networking, it is sufficient to help differentiate what will and will not be considered further in this thesis.

In order to proceed, several assumptions must be made:

- Web browsers are largely similar and run at a high level of efficiency due to their maturity and the collaborative effort and investment in their creation.
- Web browsers all have the same core functionality, primarily based on W3C, WHATWG, and ECMA standards. As a result of standardisation, browsers all perform an almost identical task, with mildly different optimisations and user experiences.
- The lower levels of networking within the system function correctly and without error. For example, devices will always be discovered, security will always be perfect, and messages will always be successfully transferred. While these are each very significant and unresolved issues, they are outside the remit of this thesis.

These assumptions do, of course, ignore many current issues in the IoT, yet they enable abstraction to a higher level. For the purposes of this thesis, they are someone else's problem.

3.5 Framing of the Analysis

The analysis within this thesis will use a set of analytical tools to assess how well the solutions meet their respective aims. As the aims are both technical and social, the approaches to each must be very different.

The outputs of the experiments in this thesis will be assessed by looking at them from both a technical and social perspective. The overall framework is a new creation, but the method used within each is not. The approach to each experiment will first be assessed for whether it is possible to implement, then if it is practical from an engineering perspective, and finally, whether the outcome is acceptable to and usable by the target demographic. Each of these steps mostly depends on the success of the preceding one, and each probes the solution a little further to ultimately show whether it can be considered a good solution or merely an engineering curiosity. This is important because, while the approaches used in this thesis are almost entirely unique from an engineering perspective, it is necessary to assess their wider potential as a part of their integration into the social machine of the Web.

3.5.1 Possibility

On the surface, possibility is a binary outcome. In technology this tends to be a positive one, in that anything reasonable can be done, and also that many unreasonable things can too. The test here is whether the technical aims can be implemented within the browser environment and within the confines of existing Web technologies. While it is expected that the approach will be possible, there are many technical details to be considered and worked through, and several possible implementations remain to be attempted.

3.5.2 Practicality

Looking deeper, it becomes a question of practicality, and whether the time, effort and resources are worth spending. This applies both on a macro scale for the entire project, as well as on a micro-scale down to the efficiency of storing the component of a device in the DOM. A major mitigation of this approach is the use of other people's work and indeed the approach of 'standing on the shoulders of giants' is a core tenet of modern software development.

This project is heavily reliant on existing browsers as it would be almost impossible to create a new browser from scratch. Doing so would break one of the core aims of this thesis which is to use existing technology as much as possible. It would also be redundant as the most popular browsers implement the vast majority of Web standards. A key metric of the practicality of any solution will be to what degree the browser had to be worked around or against to implement it, and how much the browser would have to change to accommodate such innovations.

Scale is another concern. While it may be that a solution is very simple to implement in the browser, it could be incredibly inefficient to do so. For example, if a solution performs worse than O(n), where n is the number of devices, it would more than likely be impractical in large-scale deployments.

Luckily, many of the issues of scale have already been answered, as the DOM has been designed to do this well. However, any workarounds, requirements or extra code required may not work as efficiently as the browsers' own code and, if a requirement of this WoT approach breaks the scalability, then it will present a problem that would need to be investigated.

3.5.3 User Acceptance

Even if an approach is absolutely possible and practical from a technical point of view this does not mean it is a good solution. One of the aims of this thesis is to create something which will enable existing Web developers to easily transition to the WoT. This transition can only be achieved if the developers can and want to use it. Therefore, the user acceptance, or in this case developer acceptance, of the solution is very important.

There are countless ways that user acceptance can be assessed, but they all fall into either user or heuristic testing [200]. User testing could involve directly asking the user about their experience or observing the user as they carry out tasks. It could happen in real-time as they carry out the task or retrospectively, and there have been many approaches developed with various benefits and deficits. Heuristic testing is performed by an expert and assesses an idea from the point of view of a user. User testing is ideal for testing systems that already exist, while heuristic testing is best used either before a system is built or within the constraints of a purely theoretical system, however heuristic testing has merits in any project.

3.5.3.1 User Testing

The methods of user testing employed within this thesis will involve observation of potential users completing tasks and questionnaires. These are well established in the field of UX so provide a good basis for evaluation. The specifics of these will be discussed within the experimental methods.

3.5.3.2 Heuristic Testing

The majority of user acceptance will be assessed with heuristic testing performed by the author, as an expert Web developer with experience in building and controlling IoT devices. This is not a perfect approach, as noted by Nielsen and Molich [201]. One assessor is insufficient as everyone has gaps in their knowledge and perception. This can be mitigated somewhat by using established sets of heuristics for the analysis, and augmenting these with the responses from the user testing as outlined above. Overall, this is enough to ascertain whether the idea is good or bad from the point of view of the developer, although some nuances will be lost.

The heuristics used here are those created by Nielsen and Molich [201]. Although the original paper was co-authored by Rolf Molich, the 10 rules in use today are commonly attributed to Jakob Nielsen as he later tested [202] and rewrote [203] them to "derive a set of heuristics with maximum explanatory power" [204]. They were developed in 1990 and 1994 and are still relevant and in common use today, even though there are many alternatives. The type of problems they can be applied to is deliberately loosely defined, and they have been used for both software and industrial product design [205]. This is ideal for the IoT, as it is a blend of both physical and virtual UIs.

Nielsen and Molich commented on alternatives to heuristic testing [201] and classified them as formal, automatic and empirical approaches. They concluded that heuristics are a good way to assess the usability of a system, as the alternatives are not always feasible. This is often true for this thesis too. Formal approaches do not exist in any generically applicable way, automatic approaches are limited, while empirical approaches are too labour intensive and are ultimately deficient in scope. While there have been many advancements in modern technology, particularly in relation to automation, the developer interface for the systems described in this thesis is sufficiently novel not to be able to take advantage of these.

Nielsen's Heuristics, verbatim from the original [204]

1. "**Visibility of system status:** The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time."
2. "**Match between system and the real world:** The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order."

3. “**User control and freedom:** Users often perform actions by mistake. They need a clearly marked ‘emergency exit’ to leave the unwanted action without having to go through an extended process.”
4. “**Consistency and standards:** Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.”
5. “**Error prevention:** Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions, or check for them and present users with a confirmation option before they commit to the action.”
6. “**Recognition rather than recall:** Minimize the user’s memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.”
7. “**Flexibility and efficiency of use:** Shortcuts - hidden from novice users - may speed up the interaction for the expert user such that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.”
8. “**Aesthetic and minimalist design:** Interfaces should not contain information which is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.”
9. “**Help users recognize, diagnose, and recover from errors:** Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.”
10. “**Help and documentation:** It’s best if the system doesn’t need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.”

Despite the widespread use of heuristics, and the analysis by Nielsen (concluding that it is usually sufficient), there is a great deal of value to be gained from direct interaction with the development community, for three reasons. First, as the experimental systems will be both built and tested by the same person (the author, who is professionally aware of the heuristics), the system will likely, to a certain degree, meet the heuristics through design. This is not inherently bad as, assuming that the heuristics are the measure for a good system and the system can be built within those parameters, then it will have at least some of the characteristics of a good system. However, an external opinion will always highlight those issues and ideas which would otherwise be missed. Second, as mentioned, one expert is insufficient for a true heuristic analysis, and finding another person willing to understand each experiment to the necessary extent, with no material benefit to themselves, is both unrealistic and unlikely. Additional user research will help to form a more holistic view, revealing either consensus or disagreement with the author's analyses. Finally, one of the central aims is that the development community can use the approaches as laid out in this thesis, and so direct feedback on this aspect will be very valuable not only in assessing the feasibility of the technical approach but also in gauging its potential for adoption.

Several other sets of heuristics have been created since the late 1980s, although none are as influential or as widely applied as Nielsen's. The only other set worth mentioning here is Schniederman's Eight Golden Rules [206].

Schneiderman's rules were the first set of published usability heuristics and were made available in 1985. They have since undergone minor revisions, yet remain, at their core, much the same. They are the heuristics on which Nielsen and Molich based their own set, so have weight as context.

Schneiderman's Eight Golden Rules of Interface Design, verbatim from the original [206]

1. "**Strive for consistency** Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent color, layout, capitalization, fonts, and so on, should be employed throughout. Exceptions, such as required confirmation of the delete command or no echoing of passwords, should be comprehensible and limited in number."
2. "**Seek universal usability** Recognize the needs of diverse users and design for plasticity, facilitating transformation of content. Novice to expert differences, age ranges, disabilities, international variations, and technological diversity each enrich the spectrum of requirements that guides design. Adding features for novices, such as explanations, and features for experts, such as shortcuts and faster pacing, enriches the interface design and improves perceived quality."
3. "**Offer informative feedback** For every user action, there should be an interface feedback. For frequent and minor actions, the response can be modest, whereas for infrequent and major actions, the response should be more substantial. Visual presentation of the objects of interest provides a convenient environment for showing changes explicitly."

4. “**Design dialogs to yield closure** Sequences of actions should be organized into groups with a beginning, middle, and end. Informative feedback at the completion of a group of actions gives users the satisfaction of accomplishment, a sense of relief, a signal to drop contingency plans from their minds, and an indicator to prepare for the next group of actions. For example, e-commerce websites move users from selecting products to the checkout, ending with a clear confirmation page that completes the transaction.”

5. “**Prevent errors** As much as possible, design the interface so that users cannot make serious errors; for example, gray out menu items that are not appropriate and do not allow alphabetic characters in numeric entry fields. If users make an error, the interface should offer simple, constructive, and specific instructions for recovery. For example, users should not have to retype an entire name-address form if they enter an invalid zip code but rather should be guided to repair only the faulty part. Erroneous actions should leave the interface state unchanged, or the interface should give instructions about restoring the state.”

6. “**Permit easy reversal of actions** As much as possible, actions should be reversible. This feature relieves anxiety, since users know that errors can be undone, and encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data-entry task, or a complete group of actions, such as entry of a name-address block.”

7. “**Keep users in control** Experienced users strongly desire the sense that they are in charge of the interface and that the interface responds to their actions. They don’t want surprises or changes in familiar behavior, and they are annoyed by tedious data-entry sequences, difficulty in obtaining necessary information, and inability to produce their desired result.”

8. “**Reduce short-term memory load** Humans’ limited capacity for information processing in short-term memory (the rule of thumb is that people can remember “seven plus or minus two chunks” of information) requires that designers avoid interfaces in which users must remember information from one display and then use that information on another display. It means that cellphones should not require reentry of phone numbers, website locations should remain visible, and lengthy forms should be compacted to fit a single display.”

There are, as expected, several overlaps with Nielsen’s list. However, Nielsen’s were selected for use here as they are far more refined and have a clarity gained from extensive field testing. Schniederman’s, in contrast, were produced by summarising the extant literature on software user interface design and this is evident in the otherwise incongruous mention of dialogs. Neilsen’s were designed to be more generalisable, as is demonstrated by the mention of emergency exits as an illustrative use case, despite their evolving from a software and hardware background. Nielsen’s business partner Don Norman goes on to expand on this generality in great detail in his book ‘The Design of Everyday Things’ [205].

Heuristic testing has some significant drawbacks as was highlighted by Nielsen in his papers. The key drawback, as aforementioned, is that every assessor has gaps in their knowledge. However, this can be mitigated by multiple assessors with varying degrees of knowledge. More specifically, it was found that experienced assessors overlooked simple issues, while inexperienced assessors, as anticipated, missed complex ones. Further experimentation determined that multiple assessors of varying skill levels will tend to be more efficacious than one truly expert assessor, a finding in agreement with the concept of the “wisdom of the crowd” as explored by Surowiecki in his 2004 book [207]. A related drawback is that the assessment is subjective in terms of what is deemed to be good or bad. Every developer will have their own experiences and opinions and this will inevitably lead to a variation in how they view and rate the system they use.

An important advantage of heuristic testing is that anyone can perform it, although the level of completeness will tend to vary with knowledge and skill. Linked to subjectivity, it is also a matter of opinion as to what level of completeness is deemed adequate. This has been mathematically explored by Nielsen and Landauer [208], who found that the number of users required for a good result is usually surprisingly low, although the threshold for any particular study remains a subjective choice.

Chapter 4

Methodology

This chapter explains the globally-applicable methodological choices made to implement the aims of this thesis in the experiments described in Chapter 5. The methodology of the experiments are described separately within self-contained sections of the next chapter. The aims of this thesis are necessarily high-level and, as such, taking them from this high-level abstraction to practical implementations required that several evaluations, refinements and decisions be made.

Treating IoT devices as elements of a Web page is quite a theoretical aim and one that can be assessed without the need to build anything. A thought experiment is all that is necessary, although an implementation is more convincing and may highlight any obvious issues. By contrast, building a system for controlling IoT devices that uses browser technology is a practical experiment and one that requires a working prototype to demonstrate its feasibility. Finally, exploring acceptance is less technical, yet requires critical evaluation and the opinions of others within an experimental setting. These complementary, yet different scenarios are explored in Experiments 1 through 4 in Chapter 5.

4.1 Hierarchical Modelling of Physical Environments

Physical environments are not typically modelled as a hierarchy or tree. They are often described as a list of objects, or else visually represented as a 3D model. The use of a hierarchical structure to represent a physical environment is not without its flaws, but provides many benefits that are demonstrated in the following chapters. This approach, called a compositional hierarchy, is used both in computer vision [209] and object-oriented programming [210] and, as such, is frequently recommended to user experience and digital designers [211] and so will align with how many Web developers see the world. It also suits the construction of a built environment, as we first build a room and then fill it with furniture. Then we fill that furniture with other objects in turn. However, how we perceive and reason this environment on a day-to-day basis is beyond the scope of this thesis.

Compositional hierarchies, abridged to ‘hierarchies’ from here on, are employed in this thesis as they are relatively simple to understand, particularly to a developer, while not inherently dictating the semantic context of the relationships of their contents. The relationship between two objects in the tree may be based on their physical positions or something else, such as ownership or connectivity through a protocol. There are undoubtedly situations where this approach does not work, including where an object has multiple parents. This could be, for example, choosing to model a desk as a tabletop placed on multiple legs. This can also often be solved, for example by making the legs into children of the tabletop, or by creating a pseudo-parent that contains the legs and top.

Hierarchies, as a tree structure, come with potential benefits over a list, which is the obvious alternative. A list is simple but does not allow for semantic ideas such as ‘contained within’, ‘part of’ or ‘owned by’ which can be implied by a hierarchy. For example, a list of devices as in Figure 4.1 compared to a hierarchy of those same devices in Figure 4.2. The hierarchy shows that the bulb is a part of the lamp and that the lamp is on the table. While the relationship is not explicit, it can be easily inferred by a user. A hierarchy is particularly powerful when used with virtual elements to group devices. In the case of Figure 4.2 this is demonstrated by the ‘Music System’ element. A tree structure also brings with it efficiencies in relation to search; for example when you are searching for a lamp and you know that it is on a desk. In this case, the lamp could be found by inspecting 5 elements within the hierarchy in Figure 4.2, yet would take 11 inspections in an ordered traversal of the list in Figure 4.1. These are important reasons as to why a hierarchical modelling approach was chosen over a simple list.

```
- Desk
- Bulb
- Monitor
- Laptop
- Mouse
- Keyboard
- Amplifier
- Left Speaker
- Right Speaker
- Chair
- Lamp
```

FIGURE 4.1: A list of IoT devices.

```
- Chair
- Desk
    - Laptop
        - Mouse
        - Keyboard
    - Music System
        - Amplifier
        - Left Speaker
        - Right Speaker
    - Lamp
        - Bulb
    - Monitor
```

FIGURE 4.2: A hierarchy of IoT devices with a single virtual music system element.

4.2 Using Web Technologies to Model Environments

When building a WoT system, it seems logical to use as many of the technologies that form the Web as possible, and while Web protocols such as HTTP/2 [110] are widely used in other projects, it would seem that the DOM, HTML, XML, and CSS are under-utilised. As a result, the Web browser and its attached optimisations have hitherto been mostly ignored. In a browser, the DOM [9] is used to model elements of a Web page and their physical interrelationships. This project reuses the DOM in multiple ways to model IoT devices and their resultant interrelationships. With this comes the ability to make CSS a part of the WoT, and to use JavaScript as it is used in a browser.

The DOM and HTML specifications allow an element to be stored in multiple ways. They could be stored using existing HTML elements (e.g. a `<div>` or `<section>`) with some kind of identifier to show how they are connected to an IoT device (e.g. Figure 4.3). However, it makes more sense to use a semantic element of some kind, as shown in Figure 4.4. This figure shows two alternatives that were explored separately. The first, an XML-based system which used `<device>` elements was explored in Experiments 1 and 2. These are valid XML but not valid HTML. Later, the HTML Custom Element specification [32] became more prominent and gained near-universal browser support and so a new, more browser-compliant system was built in Experiments 3 and 4. Figure 4.4 shows the syntax used for each, the key visual difference being that HTML Custom Elements require a '-' in the name. HTML Custom Elements are much more useful to a Web developer as they can be included within existing HTML documents and so can seamlessly create hybrid data/IoT documents.

```
<div device-id="14529f74-4d8b-4e43-abb1-384101819e75"></div>
```

FIGURE 4.3: A `<div>` element linked to an IoT device.

```
<device />
<iot-lamp />
```

FIGURE 4.4: Two examples of bespoke elements, an XML element (top), and an HTML Custom Element (bottom).

4.3 Storing Device State

Once the DOM can reliably represent the relationship of IoT devices, the question of representing the devices themselves can be explored. There are many options when it comes to storing data about an IoT device within the DOM, and digital twins have become the industry standard. This thesis takes a slightly different approach to digital twins from other implementations. Often, the digital twin is stored as a self-contained object, with its affordances, properties and state defined within this object. Experiment 1 follows this approach fairly closely, with the state stored in the XML element as attributes; however, Experiments 2 through 4 diverge from the standard. Despite this, the implementations are not at odds with the overall progression of the Web, and thus remain somewhat compatible with the W3C WoT Group's approach of using Thing Descriptions as a template for WoT devices. This could be achieved using a reference, as seen in Figure 4.5.

```
<device twin="https://example.com/device\_11232" />
```

FIGURE 4.5: An example of referencing data for an IoT device in the DOM.

Experiments 2 to 4 use custom CSS properties to represent state and store that state externally to the element. This allows for some level of de-duplication and is aligned with how the visual state of HTML elements is stored. This marks a key step in progressing towards an implementation that is more closely aligned with the workflow and approaches that Web developers use to build the Web.

4.4 Linking to Devices

Only Experiment 2 involves connections to physical and virtual devices, however, the approaches of the other experiments also have the potential for the digital twins to be linked to the devices they represent. Experiments 1 and 2 assume a 1:1 mapping of a digital twin to a device and so any changes to either the device or the twin itself would be copied to the other. Experiment 2 puts this into practice through a complicated event-based messaging system.

The system built for Experiments 3 and 4 differs because it does not presume this 1:1 relationship. Instead, the device and its multiple twins are all treated as equals, and thus any change to one of them would be propagated through the system until they all reflect the same, correct state. This is more representative of a Web 2.0 system, whereby a user may send a message to a channel and this message propagates to the other users, either directly or else via a server.

4.5 Assessment Strategies

The experiments were designed to assess the three potential approaches identified during the course of this project. Each approach and experiment grew from the previous one, and added something extra to the overall picture. They were all assessed using the methods outlined in the previous chapter, although to varying degrees and with differing foci.

Initially, it was important to know that the DOM could be used, and so Experiment 1 was a largely theoretical exercise. This assessment required heuristic analysis, which helped to find the flaws in the approach and pave the way for Experiment 2. Experiment 2 was performed with the knowledge that the implementation was possible and so focused more on its practicality. This led to the construction of a working Web-based IoT control system, one which included physical and virtual devices. The issues in the assessment of practicality, as well as technological advances, helped to specify the system used in Experiments 3 and 4. Experiment 3 was the closest to a browser-based WoT system, yet lacked a full physical implementation and so, once again, a heuristic analysis was chosen to assess it. When this approach was found to be adequately acceptable, Experiment 4 was devised to engage experienced Web developers who would be the target market for such a system and thereby probe its acceptability. The following chapter describes and analyses the experiments both separately and in detail.

Chapter 5

Experiments

5.1 Experiment 1: Using Attributes to Store State

It is generally encouraged that parameters relating to the state of an element that need to be stored within the DOM will be stored as attributes of the element itself. The HTML Element specification [4] demonstrates this, and the newer Custom Elements specification [32] also relies on attributes and properties for passing data to a user-created element. An example can be seen in Figure 5.1.

```
<img src="https://localhost/images/graduation.jpg">
```

FIGURE 5.1: A 'src' attribute on an image element, which shows the location of them image to be displayed.

The initial implementation of a WoT system employed this recommended approach, and used XML to allow for the assignment of arbitrary element tag names such as `<device>` and `<component>`. This was conceived before there was widespread support for Web Components, so was the optimal choice. HTML was initially more restrictive and only allowed a short list of Web-related tag names [212]. With HTML5 and the Custom Elements specifications this has since been relaxed, allowing for arbitrary tag names in the former and elements with internal processing and semantic meaning in the latter. The initial plan was to use the DOM purely as a data structure to hold the environment, rather than as a part of a document as delineated in later experiments.

A device could be represented using a `<device>` element. Then, any state was stored as attributes on the device, as in Figure 5.2. A device was placed within a `<context>` element and could have children which were `<components>`. Each of these could have data stored in its attributes.

```
<device type="lamp" power="on" color="red"></device>
```

FIGURE 5.2: A device element with state.

The result was a DOM represented in XML, wherein each element was a device or else a container for devices. The element could contain state, such as temperature or brightness, again stored as attributes on the element. The approach not only allowed for representation as a simple list, but also as a tree, which inspired subsequent experiments. Each device was a digital twin that could be linked to a real device via a `link` attribute. An example tree is shown in Figure 5.3.

```
<context name="living_room">
    <device name="television" link="https://example.com/device" power="on">
        <component name="power_light"></component>
    </device>
</context>
```

FIGURE 5.3: An example XML tree of IoT devices, with state represented using attributes.

Initially, this approach was designed to use only XML and JavaScript, but it became the inspiration for using CSS as a state store and control method in later experiments.

5.1.1 Discussion

It was realised early on that this approach could be significantly improved upon and so it never reached the implementation stage. This was for several reasons relating to practicality. However, it is included because it played a large part in inspiring the future direction of the project and is the first example of using a DOM to represent an environment. It is also foundational to the approaches used later in this thesis.

5.1.1.1 Possibility

The concept of using JavaScript to manipulate XML is well established and JavaScript is designed to work with XML in the same way as HTML. The system behind it would have been very much the same as the one in Experiment 2. There are no open questions as to whether it could have been done and it only failed to be implemented when it was considered whether it should be done at all.

The thought experiment shows that the environment can be represented in a DOM and, further, that this environment could be extended to nearly any practical size, as XML can comfortably accommodate millions of elements; the actual number being limited only by the parser.

5.1.1.2 Practicality

Ultimately, the experiment was cut short because it would not have scaled well or integrated effectively with other Web technologies, despite following a standard approach.

Scaling was an issue because of the hierarchical nature of the tree structure. There were two options for storing the state of a component within a device wherein that state also depended on the parent device (e.g. a power state). Either the state would be set on the device (as in Figure 5.4), or on the device and all children (Figure 5.5).

```
<context>
    <device power="on">
        <component></component>
        <component></component>
        <component></component>
    </device>
</context>
```

FIGURE 5.4: Setting the state on just the parent.

```
<context>
    <device power="on">
        <component power="on"></component>
        <component power="on"></component>
        <component power="on"></component>
    </device>
</context>
```

FIGURE 5.5: Setting the state on the parent and all children.

The first is optimal for writing as it requires changing a single attribute, yet is inefficient for reading as a component as it would have to read its own state and the state of all parents, which is O(log n) (where n = number of nodes under a device). It would also present a confusing user experience as, similarly to HTML and CSS, not all states should be inherited by children, and not all states of a parent would apply to a child.

The second is optimal for reading as it would only require reading a single attribute, yet is inefficient with respect to writing and the storage size of the DOM, given that the attribute would have to be applied to a parent and all children which is an O(n) operation (where n = number of nodes under a device). It has a mildly better user experience as it is explicit about the state of each element within the DOM. While O(n) is not usually considered a bad outcome, and O(log n) is often considered good, neither can take advantage of the browser's built-in optimisations, or use its native code, as the inheritance of attributes is outside the DOM specification. Additionally, applying a state to multiple devices would always be an O(n) (where n = number of devices) action as every device would need to have an attribute created or updated. In contrast, changing the state purely in CSS – ignoring rendering – is almost always an O(1) action as the state is updated in a rule rather than within the elements themselves.

The approach also did not integrate well with other Web technologies. Using XML immediately excluded a large number of existing libraries which were designed only for HTML. This would have created a large barrier to adoption and acceptance by the community. During the experiment it became increasingly obvious that the use of CSS as a selection language could provide many benefits over the more traditional XPath used for XML and, ultimately, pursuing this path rendered the attribute-based approach increasingly obsolete. It was this decision that was the overriding reason that this experiment was not implemented in code. While CSS can work with XML, the selection approach used by CSS is heavily targeted towards the IDs and classes in HTML. Using the attribute selector (e.g. `device[power="on"]`) to select all devices that are currently turned on was considered, but this approach is clumsy and works around the problems within the approach, rather than working with the language and the established industry standard approaches for the application of CSS.

The one advantage of this approach over any of the later ones is the ability to select devices based on their state. As in the previous example, being able to select all devices that are currently powered on could be a useful feature, but ultimately is only made possible due to the state being available within the device's attributes.

Had this approach been continued, it would have resulted in an inefficient XML approach which would have worked, yet remained very verbose, both within the XML and the CSS selectors. However, the idea of using a DOM and CSS selectors together seemed possible and sensible, albeit in a way that is more conducive to industry standard approaches, and building as the specifications intended. Using the DOM with CSS selectors would allow the system to take advantage of native browser methods such as `querySelectorAll` to select all devices, components, or any subset of them. This was incredibly useful and powerful as the time and money which has been invested in Web browsers over the last 30 years has led to highly optimised renderers and selection engines that lie far beyond the capabilities of any that a solitary developer can produce. The adoption of CSS selectors naturally led to the use of HTML over XML and this, in turn, opened up the possibility of using other CSS features. However, this approach was not directly compatible with the work in this experiment and so a line was drawn and a new experiment started.

5.1.1.3 User Acceptance

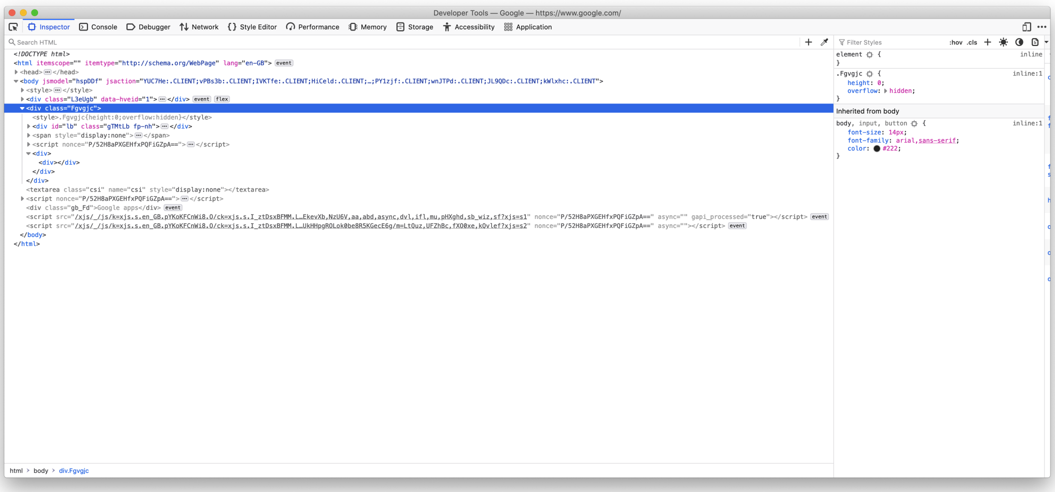
Heuristic Analysis As the experiment never progressed to the implementation stage, the best approach for assessing it in any detail is a heuristic one. Applying each of Nielsen's heuristics in turn, the system can be examined and the good features allowed to persist while flaws are analysed. This led to the following analysis, deconstructed by heuristic:

1) Visibility of system status The visibility of status is a key benefit of using the DOM to represent devices and environments. In this initial version there are two options for showing status. Either the current state is replicated on each component and device, or else the device inherits its state from its parent. In the former, the state of anything in the DOM is very visible as a developer can read the state from any device or component in isolation. In the latter, it is a little harder as the developer must know the position of the component within the tree and work up to the root to determine any state which may be inherited.

There is also a level of conflict resolution that must be overcome. Typically this would be that the child state overwrites the parent state in the case of a conflict, although there are exceptions in the modern Web. A notable exception is the use of `!important` in CSS, which elevates the value to a higher position in the evaluation process.

In all cases, an expert user can immediately look at any component and see its state, for instance whether it is turned on or what colour it is. In this respect, the approach is a resounding success.

Both alternatives have an issue of verbosity to different degrees, and this makes the raw XML much harder to parse visually than a document without multiple attributes per node. This can be overcome using tools which are designed to help the user to traverse a DOM tree. For example, the developer tools present in most modern browsers which only show the section of the tree the user is interested in and folds away the rest, as seen in Figure 5.6.



```
<!DOCTYPE html>
<html itemscope="" itemtype="http://schema.org/WebPage" lang="en-GB"> event
 ▸<head>…</head>
 ▾<body jsmodel="hspDDf" jsaction="YUC7He:.CLIENT;vPBs3b:.CLIENT;IVKTfe:.CLIENT;HiCeld:.CLIENT;…;PY1zjf:.CLIENT;wnJTPd:.CLIENT;JL9QDc:.CLIENT;kWlxhc:.CLIENT">
   ▸<style>…</style>
   ▸<div class="L3eUgb" data-hveid="1">…</div> event flex
   ▾<div class="Fgvgjc">
      <style>.Fgvgjc{height:0;overflow:hidden}</style>
     ▸<div id="lb" class="gTMtLb fp-nh">…</div>
     ▸<span style="display:none">…</span>
     ▸<script nonce="P/52H8aPXGEHfxPQFiGZpA==">…</script>
     ▾<div>
        <div></div>
      </div>
    </div>
    <textarea class="csi" name="csi" style="display:none"></textarea>
   ▸<script nonce="P/52H8aPXGEHfxPQFiGZpA==">…</script>
    <div class="gb_Fd">Google apps</div> event
    <script src="/xjs/_/js/k=xjs.s.en_GB.pYKoKFCnWi8.O/ck=xjs.s.I_ztDsxBFMM.L…EkevXb,NzU6V,aa,abd,async,dvl,ifl,mu,pHXghd,sb_wiz,sf?xjs=s1" nonce="P/52H8aPXGEHfxPQFiGZpA==" async="" gapi_processed="true"></script> event
    <script src="/xjs/_/js/k=xjs.s.en_GB.pYKoKFCnWi8.O/ck=xjs.s.I_ztDsxBFMM.L…UkHHpgROLok0be8R5KGecE6g/m=LtQuz,UFZhBc,fXO0xe,kQvlef?xjs=s2" nonce="P/52H8aPXGEHfxPQFiGZpA==" async=""></script> event
  </body>
</html>
```

```
element {                                inline
}
.Fgvgjc {                                inline:1
   height: 0;
   overflow: ▸hidden;
}
Inherited from body
body, input, button {                    inline:1
   font-size: 14px;
   font-family: arial,sans-serif;
   color: ● #222;
}
```

FIGURE 5.6: Firefox developer tools showing a folded DOM tree.

Timeliness of feedback is beyond the scope of this implementation as it would be almost entirely dependent on the connection latency arising between the user and hub and also the hub and its associated devices.

2) Match between system and the real world The system mirrors the world, but it shows only one interpretation at a time. The DOM is a hierarchy, but this could be positional, based on ownership or even arrange in a domain-specific way; for example, it may be the best option to arrange all devices into colour groupings rather than by where they physically sit in a room, the limitation being that, once the devices are in a DOM, their interpretation is fixed until the DOM is rearranged. Due to the nature of digital twins, it would be possible to have multiple DOMs representing the same set of devices or environment and this is discussed further in Chapter 7.

As mentioned previously, a compositional hierarchy is a familiar structure to developers, particularly Web developers. The usage here is a reasonable way to model a physical space for this audience and so, from that perspective, the DOM is an appropriate metaphor for describing the problem.

In terms of the appropriate use of words and phrases, the recycling of technology brings with it the re-use of terms, and those terms relate to the same concepts in this implementation as they would within any other XML document. This is one of the key benefits of this approach as it avoids new jargon or mental models.

3) User control and freedom The control of a system described in this way is solely in the hands of the developer. They are free to make changes and reverse them, and the process for each is the same. While there is no handholding in the form of a record of changes they can simply ‘undo’, there is nothing in the architecture that prevents the tool they are using from implementing it. Beyond this, the ability to control a device is identical to that of any other IoT system.

4) Consistency and standards This approach is designed to follow Web standards, yet there are a lot of different standards that could be applied to solve the problem. In this case, the standards selected are not ideal for the reasons explained above. Overall, the system would be very standards-compliant, meaning that a developer should be able to understand it with only minimal training, although this alone does not make it a good system.

Consistency is challenged when considering the human aspect of the system. Where synonyms, homonyms and different languages are considered then a lot of issues are raised. This is discussed in far greater detail in Chapter 7.

5) Error prevention Some errors can be prevented by design, because an incorrectly formatted DOM will not render in a browser. Errors should also be highlighted during any manual editing in the developer’s IDE, provided the syntax is accounted for. From a logical standpoint, there is nothing presented so far that would actively prevent logical errors. This is a problem across the development field and one which is usually mitigated by warning and error messages aimed at the developer.

6) Recognition rather than recall This is an approach in which this system could excel. As it uses Web standards and presents the environment in a format that all Web developers would recognise and understand, it scores very highly here. However, it falls down where any XML-based system would, because while the structure is very recognisable, the set of attributes used is an open taxonomy and thus the specific attributes have no semantic meaning attached to them. This is contrary to HTML, in which most attributes, such as `src`, `href` or `class`, have meaning attached to them. In this case, a developer would need a reference as to what `power` means with respect to the device it is attached to. They would also need to know which attributes are applicable to each device so as to save adding too many attributes resulting in both unexpected results and a bloated DOM. This aspect would be entirely dependent on recall, which is a strongly negative feature.

7) Flexibility and efficiency of use The flexibility of this approach is very good, as the DOM can be reconfigured in any way that is desired, and attributes can be freely attached to any element. Its downfall may be in having too much flexibility in some cases. As mentioned above, a developer could add too many attributes to an element which in turn could create unintended consequences, or else present a confusing document to the next editor. Overall, the approach is designed to be as open-ended as the Web is, to allow developers to implement it across any environment and integrate it within any pre-existing system.

8) Aesthetic and minimalist design The choice of XML and its associated attributes here is not minimalist and if the state is not inherited then it is quite the opposite. It would be very likely that a complex device with many sub-components would have a lot of duplication of state replicated from the parent to all its children. Using an inherited state would be more minimal, but having both structure and state in the same document is a fairly cluttered approach.

9) Help users recognize, diagnose, and recover from errors Any critical errors or syntax errors would be reported in the browser's console or the IDE during development, although logical errors would be hard to find. The document would be very densely laden with attributes, and finding one that is missing, or noticing an incorrect (or additional one) would be hard, due to its highly repetitive nature. Specialist tools would be needed as currently the best help a developer would have would be syntax highlighting in their IDE.

10) Help and documentation Specific documentation does not apply to this experiment as it was not implemented. However, as it follows Web standards, then the same documentation for those standards would apply here too. While developers would need basic documentation to learn how to represent an environment within a hierarchy, they should already be familiar with all of the technology used here. There are also excellent alternative sources of documentation to the official specifications which are available around the Web for learning about Web standards and how to apply them.

5.2 Experiment 2: A Centralised, Hub-Based, HTML5 and CSS WoT System

The second approach was a practical experiment that resulted in the creation of a JavaScript-based hub application, several client libraries, and some physical devices built to demonstrate it. The system was run in a real environment of varying complexity for several months. This section is focused on the choices made, their impacts, and what was learned from the implementation.

This experiment intended to take what was learned from the previous experiment and apply it to building a new system that would address some of the shortcomings, while also testing some new ideas in a deployment to a real environment.

The aims were to:

- Build a working physical prototype of a WoT system in JavaScript;
- Continue using the DOM to store the environment representation;
- Use CSS selectors for deciding which devices to route messages to;
- Use classes and IDs on DOM elements and CSS for storing state.

The test environment chosen for the experiment was a bedroom. This was an easy environment to access and monitor, but was also a relatively private space with one primary user. This reduced complexity, but also allowed for a better evaluation of the system from having to live within the experiment for many months.

A big advancement inspired by Experiment 1 was the the use of style sheets to represent state. This is an example of a DRY (Don't Repeat Yourself [213]) approach to code and separates the representation of the environment from its state. It solves the issue in the last experiment where the state required either storing many repeated attributes or retrieval required traversal up the tree. It does this by replacing attributes with style rules which are applied through CSS selectors, as demonstrated in Figure 5.7. The figure also demonstrates the ability to use the CSS cascade to override state. This solution directly parallels the use of style sheets on the Web which serve to prevent repeated `style` attributes on HTML elements and separate the visual state from the structure and its data. This approach is in line with the ethos of this thesis, namely that a lot of IoT problems could be solved using existing solutions.

The system stores the application state in the same way as any other JavaScript application, using a combination of memory, databases and configuration files. However, this state is distinct from the environment state and can be disregarded for the purposes of this experiment. The important state is that which is stored in CSS and is used to describe the

Attributes:

```
<device power="on" color="red"></device>
<device power="on" color="red"></device>
<device power="off" color="red"></device>
<device power="off" color="blue"></device>
<device power="off" color="blue"></device>
```

Classes:

```
<device class="class_a"></device>
<device class="class_a"></device>
<device id="device_x" class="class_a"></device>
<device class="class_b"></device>
<device class="class_b"></device>
```

```
    .class_a {
        power: on;
        color: red;
    }

    .class_b {
        power: off;
        color: blue;
    }

    #device_x {
        power: off;
    }
```

FIGURE 5.7: A comparison of attributes and classes.

state of the environment. In particular, the CSS describes the state of the digital twins within the environment. This is a departure from the normal use of CSS within a document, which describes style – that is the visual appearance or visual state – only. In this system, the CSS describes the complete state of the device although, with some creative thinking, this state could be considered to be akin to the style of the digital twins.

The working prototype for this approach used an HTML5 DOM tree stored inside a PhantomJS [214] instance within a NodeJS [80] application that was developed specifically for this project. PhantomJS was a headless implementation of the WebKit [215] browser which was, at the time, the basis for Google Chrome and Apple Safari browsers. The DOM was XML-based, although very similar to XHTML. The main difference was that it contained a container root node called `<context>` which replaced the `<html>` node which would normally be present. In all other respects it was treated as an HTML document rather than an XML one.

The DOM was held within a central hub, which also housed the Web-based UI and other components that were convenient to place there. The system was designed to be open and devices were free to connect to the hub and register themselves, forming a star topology. Once registered, they were assigned a node in the DOM which would become their digital twin. From then on, anything that happened to the digital twin was communicated to the device, and vice versa. This meant that the two would be synchronised as much as possible.

Once registered, devices could register their own components, which would also be created in the form of nodes within the DOM. For example, a refrigerator could register itself, and then the light and compressor inside it. A key benefit here is that the components could be referenced and controlled individually, and even combined to form new virtual devices, a concept which is explored further in Section 5.2.1.6. This concept of Virtual Device Composition was explored by Schuster et al. [216], where composed devices are built from the parts of other physical or virtual devices. For example, a multi-room hi-fi system from several independent speakers and an audio source. It is also similar to the idea of service and device composition demonstrated by Merabti et al. [217], however they did not consider the components within a device.

After the DOM was populated, developers were free to interact with it using various commands sent to the hub. These were limited in scope, yet powerful. They are discussed further in Section 5.2.1.7. The hub would act upon these commands using CSS selectors and built-in browser functions or to add a new rule to the internal style sheet. The use of CSS selectors allowed for the complex targeting of nodes using the set-based concepts laid out in the aside in Section 2.2.2.2.

The rules in the internal style sheet contained properties such as `power`, `position` (with x, y, z measurements), and `color`. The syntax of this language was the same as for CSS, but browser CSS processors have an allow list for properties that are valid CSS, so the hub application had to have additional code to replicate and replace some functionality. This could have been overcome with CSS Custom Properties, although they were not yet available in most browsers, including PhantomJS. The rules were used in the same way as CSS, but rather than rendering them on a Web page, they were instead ‘rendered’ in an environment.

The system did push the concept of using front-end Web technologies a little too far. It experimented with using CSS selectors for routing messages, an approach that would seem in of itself to be a novel idea, but ultimately not overwhelmingly successful. The failings of this are discussed later in this section.

The initial version of this system was published in the Late-Breaking Work section of CHI 2019 [218], but it developed significantly after that extended abstract was written.

5.2.1 Technical Implementation

The system was composed of a central hub device running a JavaScript server application (the server). A client library – nicknamed ub.js – was written in JavaScript and carefully designed to be universal across all the hardware platforms used. This was embedded within several devices and Web applications (the clients). The server ran NodeJS while the clients were running either in various Web browsers, on other NodeJS based servers, or on constrained hardware devices using an embedded JavaScript implementation called Espruino [81]. There was also a failed attempt to use an Arduino [219] running a client in Arduino C.

The clients communicated with the server using a variety of protocols, including Web Sockets and REST, and also over a serial port. The specifics of these implementations are of a lower level than is relevant, although the diversity demonstrated that the system was viable in a real environment with real heterogeneous devices.

A key feature was that the clients had no knowledge of the rest of the system. They did not need to maintain a list of other devices, as they simply had to send a message to the hub and this message would either match devices or it would not. The developers of the clients would need to know whether the device classes were present within the environment, and could optionally know the IDs of specific devices, but ultimately they would not need to know whether other devices were present, or how many of them were available.

5.2.1.1 The Hub

The hub was central to the system and handled connection to devices, the hosting of services, and contained the DOM. The hub's DOM was the source of truth for the system and this truth was reflected by the devices connected to it. Practically, this meant that when a client or device requested the state of another device, the hub returned the value from its DOM, rather than querying the device itself. This assumption was fine for this system, yet was likely not to be acceptable for safety-critical systems or those that might pose a threat or danger (e.g a gas oven). In this case it would be potentially catastrophic if the DOM did not accurately represent the state of the device.

The Web-based UI was hosted by the hub as it was an efficient use of hardware, although it could have been hosted elsewhere. The Web UI was itself a client of the system once loaded by the user's browser, but needed a basic Web server to distribute the static files. The hub also contained virtual devices and composed devices, although this was again a matter of practicality rather than a constraint of the system, as they could have been hosted anywhere. An overview of the complete architecture can be seen in Figure 5.8.

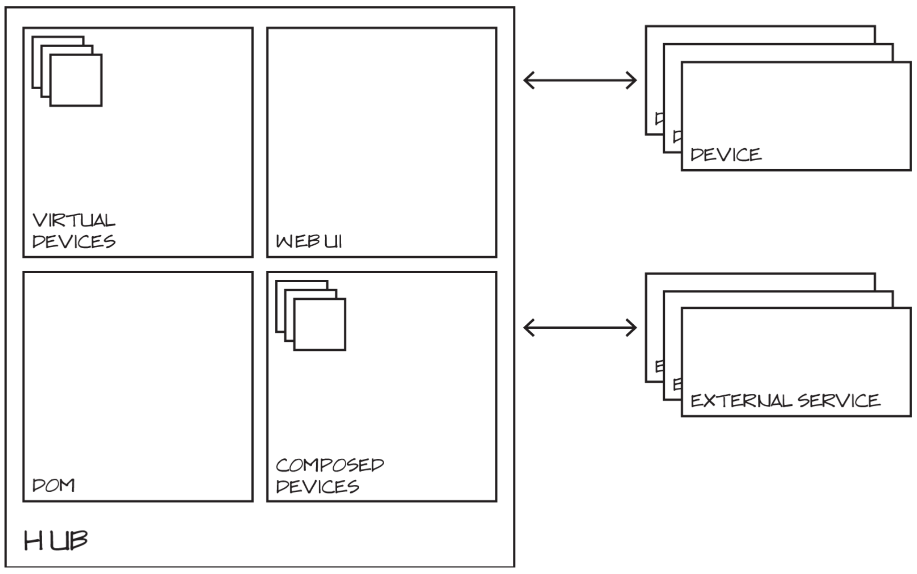


FIGURE 5.8: The architecture of the hub.

The hub had a basic but powerful API for performing functions on the DOM. It allowed clients to:

- Register themselves and their components
- Read the state of a device or devices matching a CSS selector
- Update the classes or ID of a device or devices matching a CSS selector
- Update the whole digital twin for a device matching an ID
- Remove a device from the DOM
- Send a new rule to be added to the hub's internal style sheet
- Delete a rule from the style sheet

A New Publish/Subscribe Model

Additionally to the DOM API, the hub also had an API for a publish/subscribe model based on the DOM. This allowed devices to create and delete channels, which were created as nodes in a separate DOM. Each channel could have and ID and classes. Devices could then subscribe or unsubscribe to these channels using a CSS selector to select the channels they were interested in. For example, subscribing to `.temperature` would subscribe them to all temperature channels in the system, and there was a built-in feature to subscribe them to all future matching channels using a style sheet to store their preferences. A subscription was simply adding the device as a node in the DOM under the channel parent node. Once inside a channel, the devices could publish messages which were received by the hub and forwarded to all children of the channel node.

This structure enabled devices to subscribe to multiple channels in a very flexible way. The result was a publish/subscribe model which far surpassed MQTT's wildcard `room/+/temperature` approach and allowed for an unknown amount of complexity in the structure. This was outside the final scope of this thesis, but was an interesting and related application of Web technologies nonetheless.

5.2.1.2 The ub.js Library

The library which formed the core part of each client was nicknamed ub.js. It was designed to be familiar to Web developers and, as a result, it was decided that it would use a jQuery-like syntax [220] rather than a new API structure.

jQuery is a library that was released in 2006 and designed to give a universal interface to the DOM across varying browsers. At the time there were large differences in how the major browsers implemented Web standards and how much of each standard they implemented. jQuery used JavaScript to wrap these inconsistencies with an API that was the same across all browsers and, more importantly, produced consistent results. While it is not used as widely now due to improved consistency across the major browsers, it is still familiar to most Web developers.

ub.js uses a similar API to jQuery and developers can use it to select devices within the hub's DOM in much the same way that they select DOM elements on a Web page. For example, `ub('#lamp')` selects the device with an ID of 'lamp', while `ub('.light.blue')` selects all devices with the classes of `light` and `blue`. The selectors here are CSS selectors in both jQuery and ub.js. Modern browsers have a similar interface which uses `document.querySelectorAll('.light.blue')`, although this was only implemented in 2009. It was also hard to override built-in browser functions on HTML elements until the Web Components specification was fully implemented, so jQuery added a simple interface for manipulating these.

Once a developer had chosen a selector, they could then call upon a number of functions on devices or device components matching that selector using ub.js. These included:

- Applying a CSS property from an extended allow list (including `power` and `temperature`)
- Directly calling an action
- Reading the device's current state as CSS

- Updating the ID
- Updating the class(es)
- Publishing a message to a channel (or channels) matching the selector
- Subscribing to a channel (or channels) matching the selector

Additionally, the library could:

- Register a new device in the DOM
- Register a new component of a device in the DOM
- Register a new channel in the DOM
- Remove each of the above from the DOM

5.2.1.3 The Web Interface

The Web interface was a client and ran a copy of the ub.js library. However, it was designed for direct human interaction, which sets it apart from other devices in the system as these were all automated. A screenshot of the final interface can be seen in Figure 5.9. It was composed of switches, sliders and indicators which were analogous to sensors and actuators in physical devices. The first iteration of the Web interface was the first client of the system due to the relatively low temporal and effort cost of implementation as compared with building and programming hardware.

The first version of the Web interface was a virtual lamp that was rendered in a browser on the same computer which was running the hub software. It connected using the ub.js client library and appeared to the server in precisely the same way as a physical lamp would have. The fact that devices can either be physical or virtual with almost no effect on the underlying functionality of the approach is also exploited in Experiments 3 and 4. The lamp was the simplest possible device and only had a bulb and a switch, with two power states (i.e. on and off).

Two approaches were investigated to toggle the power to the bulb. Initially, the switch would toggle the bulb and then transmit the new state to the hub which was then used to update the digital twin within the hub's DOM. However, this had two issues. The first was that it was not very useful for testing the selector-based messaging and, second, it broke the assumption that the hub's DOM was the source of truth. In this case, the lamp was the source of truth and the DOM was updated later to match it. The alternative was following a more complex process, one which was more useful for the experiment:

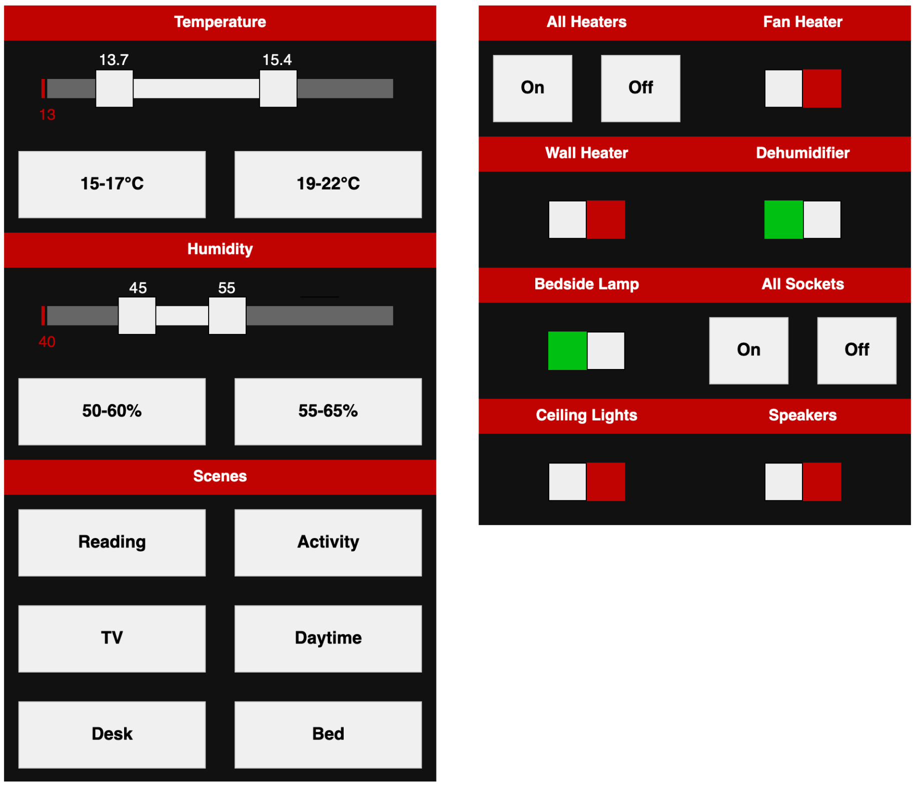
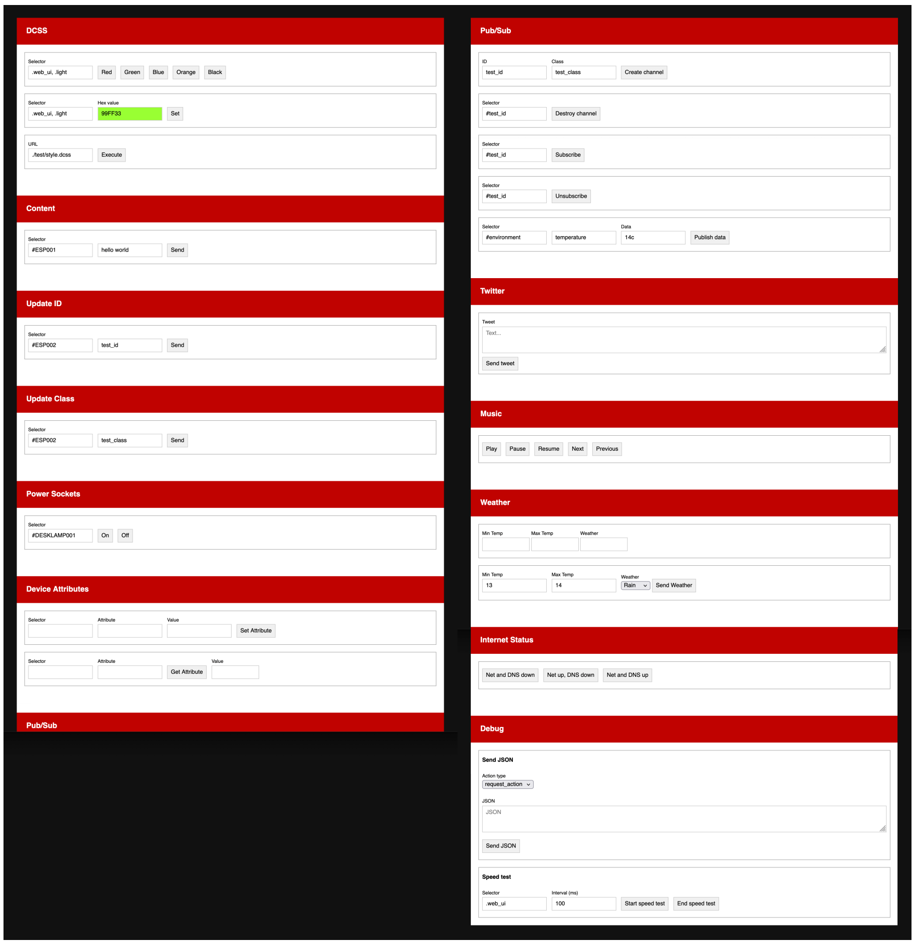


FIGURE 5.9: The Web interface.

- The switch targeted the lamp using a selector and sent a message to the hub. This message contained the selector and a command to turn the power on, i.e. `.lamp { power: 'on'; }`.
- Once the message arrived at the hub, it used the selector on the internal DOM and retrieved a list of matching devices. In this case, it returned a single device.
- The hub appended the new rule to its internal style sheet for the environment so that any new devices connected later could be 'styled' correctly.
- For each matching device, the hub forwarded the message so that the device could act upon it. In this case, only the lamp received the message.
- The receiving device interpreted the message and acted accordingly. Here, the lamp understood what `power` refers to, and understood that `on` should trigger a specific action.

This second approach treated the DOM as the source of truth, albeit with some obvious downsides as discussed later in this chapter.

After this initial success, the Web interface was progressively expanded to control other clients built for the system. By the end of the deployment it had sliders for controlling temperature, humidity and speaker volume; a colour picker for a real lamp; a text box for sending messages to an LCD screen; and various switches for speakers and mains powered devices. It also had a debugging interface for sending messages as text (as seen in Figure 5.10), and a page which showed the hub's DOM and internal style sheet.


DCSS

Selector: .web_ui, .light [Red] [Green] [Blue] [Orange] [Black]

Selector: .web_ui, .light — Hex value: 99FF33 [Set]

URL: ./test/style.dcss [Execute]

Content

Selector: #ESP001 — hello world [Send]

Update ID

Selector: #ESP002 — test_id [Send]

Update Class

Selector: #ESP002 — test_class [Send]

Power Sockets

Selector: #DESKLAMP001 [On] [Off]

Device Attributes

Selector — Attribute — Value [Set Attribute]

Selector — Attribute [Get Attribute] Value

Pub/Sub

Pub/Sub

ID: test_id — Class: test_class [Create channel]

Selector: #test_id [Destroy channel]

Selector: #test_id [Subscribe]

Selector: #test_id [Unsubscribe]

Selector: #environment — temperature — Data: 14c [Publish data]

Twitter

Tweet: Text...

[Send tweet]

Music

[Play] [Pause] [Resume] [Next] [Previous]

Weather

Min Temp — Max Temp — Weather

Min Temp: 13 — Max Temp: 14 — Weather: Rain [Send Weather]

Internet Status

[Net and DNS down] [Net up, DNS down] [Net and DNS up]

Debug

Send JSON

Action type: request_action

JSON: JSON

[Send JSON]

Speed test

Selector: .web_ui — Interval (ms): 100 [Start speed test] [End speed test]

FIGURE 5.10: The Web interface's debugging page.

5.2.1.4 Physical Clients

After the system had been validated with the virtual bulb, several real devices were built or else connected through adaptors. These included:

- An RGB LED desk lamp
- An LCD screen for displaying short messages
- A desk fan
- A dehumidifier
- An electric heater

- A temperature sensor
- A humidity sensor
- Three wireless buttons
- A Raspberry Pi hat with red, amber and green lights
- A bed side desk lamp

Most of these clients were either connected to an ESP8266 [221], an Espruino Development board, or a Raspberry Pi. Some were plugged into radio frequency plug sockets which were controlled by an Arduino-based serial to 433MHz adaptor and virtual device on the hub. This was due to safety concerns arising from modifying devices that used mains electricity. With time, the devices could have been safely modified to contain a relay, rather than the radio-controlled sockets as these were sometimes imperfect.

Due to the constraints of running Espruino on small, embedded devices, the ub.js library had to have a limited number of variables stored in memory at any one time, meaning that the stack size could never be more than a few levels deep. The specific depth allowed depended on the complexity of the functions, but was always very limited. To avoid crashing the devices, an event-based approach was used. As opposed to calling a function directly from another function, each function pushed a message to a queue, and other functions would listen to that queue and respond accordingly. In this way, even complex tasks could be achieved with a very small memory footprint, although callbacks were more complex.

Arduino: A Failed Experiment

Initially, it was planned to use an Arduino as the client due to their low cost and the high availability of compatible hardware. Ultimately, this was a failure as a standard Arduino cannot concurrently support an IP stack, the client software, and parse JSON messages, mostly due to a lack of available memory.

A client library was created in Arduino C which communicated over serial and could parse short JSON messages, although this was sufficiently far removed from the idea of the IoT that it ceased to be relevant.

After this failure the ESP8266 was chosen as a replacement device. It has a superior capacity and separate Wi-Fi chipset that handles the IP stack without detracting from the main processor. Coupled with the Espruino JavaScript virtual machine, this made it the logical choice for this project.

Figure 5.11 shows how the client library could connect to the hub, register the device, and control another device via the hub. Clients registered sensors and actuators with the hub, which would subsequently add them to its DOM. The client could then either send messages to the hub or else respond to messages from the hub. Once a message was received, the client could choose whether to act on it or forward it on to its components which may also be independently running the same client library. This loosely coupled approach is very important to the way the entire system operated, as it meant that no device needed to know about the other devices on the network, or maintain a list of active devices.

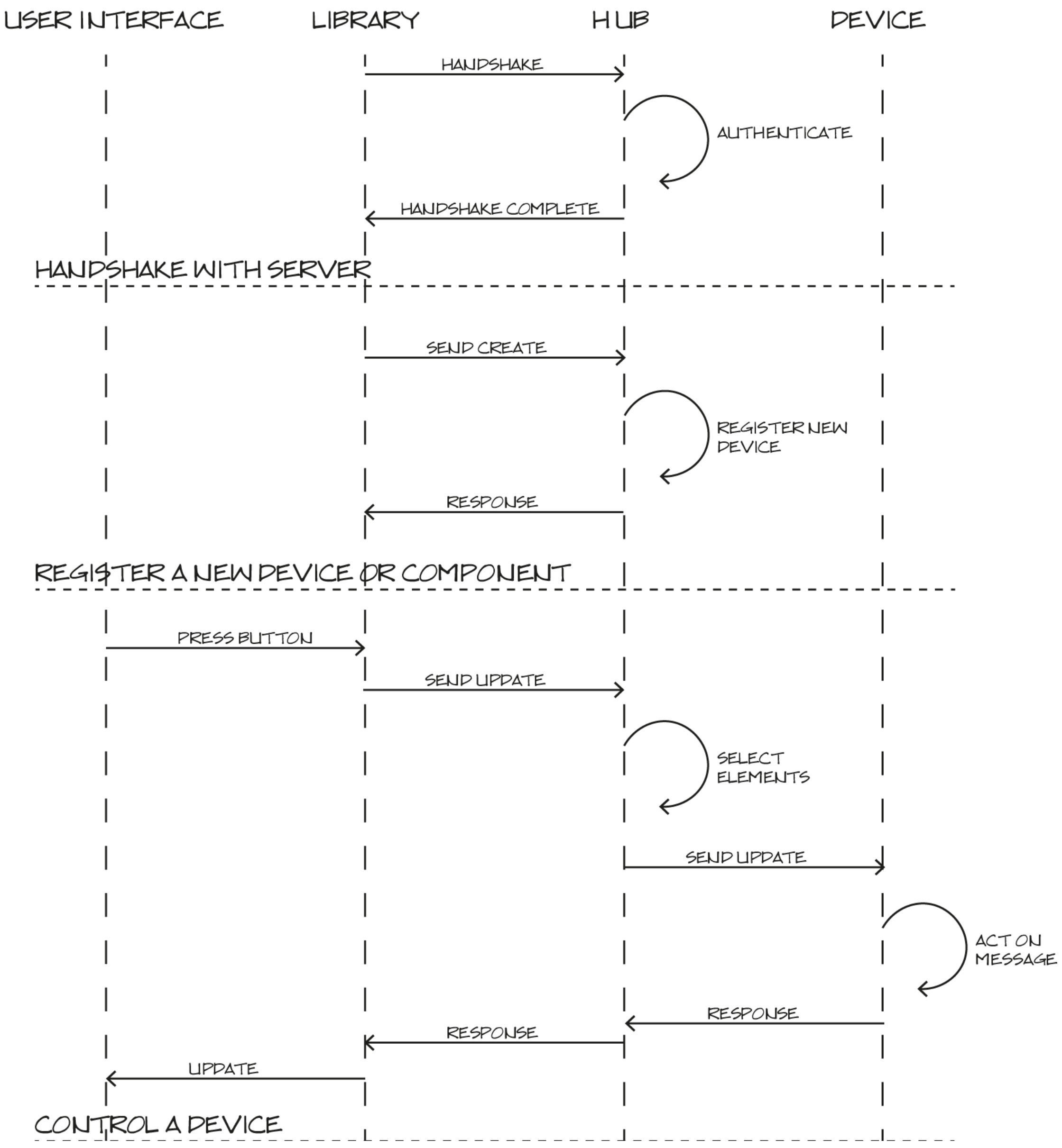


FIGURE 5.11: A sequence diagram of the key features of the client library.

5.2.1.5 Virtual Clients

Virtual clients were implemented in software and either interacted with services, other devices, or else were a proxy for other hardware. These included:

- The original Web-based lamp
- A basic media controller, sending volume and play/pause/stop commands to the OS
- A Web page that could change background colour
- A service to send SSH commands to another computer
- A service that retrieved the sunset and sunrise times
- A logging client which subscribed to channels and output messages to a file or database
- A Twitter client
- A weather service
- An Internet uptime monitor

These virtual clients allowed for slightly more complex interactions than the physical devices, as time, cost and skill are limiting factors in building complex hardware. The media controller allowed for prototyping control over a complex sound system, and the SSH controller allowed almost unlimited control over a computer, its software and connected hardware.

While some used the ub.js library and ran in their own Node.js servers, many were functions within the hub that behaved more like plugins than devices. However, the messaging approach was consistent and all virtual devices had to be represented within the DOM.

The Web page with a changing background colour was built to test the effect of having a document synchronise with hardware. In this case, the document's background synchronised with the desk lamp to provide a more immersive experience.

5.2.1.6 Composed Devices

Composed devices evolved last and through necessity. These were devices that were presented to the system in the same way as a physical or virtual device, yet were formed from the combination of two or more devices or components of devices. Composed devices have been explored previously by others [216] [217], however the method of exposing components used in this project allows for this to happen in an entirely Web-native manner.

The simplest example was a light controller, which used the time of sunrise and sunset to turn the bedside desk lamp on and off. This could also be manually overridden. It stored its state in CSS and existed in the DOM although, otherwise, it was represented in exactly the same manner as any other physical or virtual device.

A more complex and well-rounded example of a composed device in this system was the climate control system. It contained a fan, a dehumidifier, an electric heater, a temperature sensor and a humidity sensor. While it was far from perfect, it fulfilled its role of keeping the room warm and dry during winter. As this system was built in the UK, the humidity is typically too high and temperatures too low, especially as the room in question had neither adequate ventilation nor central heating. During the winter it was sufficient to turn off the heater and, due to the lack of insulation, it would mean that no cooling device was needed.

The climate controller composed device presented to the hub as any other device would. It responded to commands and gave feedback in precisely the same way as a physical device would have. However, it was entirely virtual and only existed as a concept within the system, running on the same Node.js server as the hub. It was the sum of many parts, yet no part could have individually fulfilled the requirements placed upon it. It internally monitored the humidity and temperature and stored them as CSS on the hub. It also maintained a target temperature and humidity range and used the devices at its disposal to do this. Using the fan ensured the large room had a fairly even climate. The layout of the hardware can be seen in Figure 5.12.

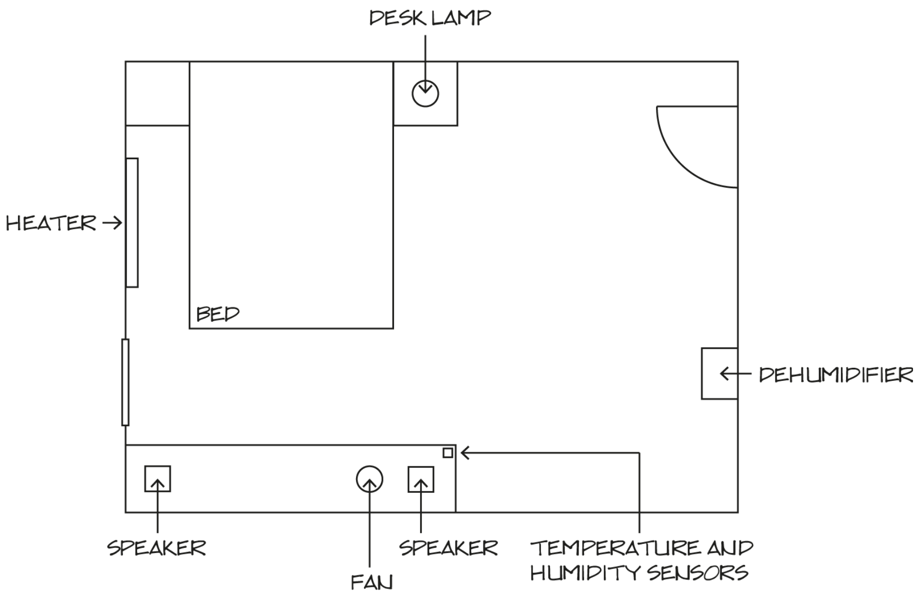


FIGURE 5.12: A plan view of the climate control system.

The system's interoperability and homogeneity lend themselves very well to building devices like this, and the level of customisation possible is unparalleled in either commerce or academia. Such a system could allow composed devices to be built automatically to fulfil specific requirements and then be torn down afterwards to save processing resources. A device could be made to send an emergency alert across all speakers and lights on every device within a building without affecting those devices of which the speakers and lights are a constituent part. Another could be built across an apartment block to search for inefficiencies in infrastructure and then destroyed once the survey was completed. Before security and privacy concerns are taken into consideration, its potential is almost limitless.

5.2.1.7 Messaging

All messages were sent as JSON objects containing an ID (in order to identify responses to messages and prevent duplication), a CSS selector (used to identify which devices to send the message to), together with a payload (containing the CSS property and new value). Figure 5.13 shows some example messages created by the system.

An obvious alternative messaging protocol would have been MQTT [113] owing to its widespread use and acceptance in the IoT. However, MQTT is too constrained for this approach and cannot adequately fulfil the requirements. MQTT relies on devices and programs publishing messages to channels and then other devices and programs subscribing to those channels to receive messages. The channels, called topics, are hierarchical and allow for wildcards, although there are several situations in which MQTT is either inefficient or insufficient. For example, CSS selectors such as `.building :not(.light)` cannot be represented in MQTT. Nor does it, by design, contain simple ideas such as `#device_one`, without creating a topic per device, or `.building .light` where there are unknown numbers of layers situated between the building and the lights. MQTT does have benefits over the CSS approach as it is a well-developed and well-supported binary messaging format, but ultimately it is not a good choice in combination with the DOM. This was also seen as a good opportunity to investigate CSS as a message routing method.

A message would be sent from the device to the hub, whereupon the hub would use the selector to choose the appropriate devices and then forward the message on to each of the matching devices. After a device had processed a message, it could optionally send a response message back to the originating device.

Messages within the hub were of the same format as messages between the hub and devices. They were sent over a global ‘bus’ which had a publish/subscribe model. All internal components and virtual devices would subscribe to the channel, and each could post messages to it. It was a practical, but insecure and inefficient method of communication.

The schemas for messages can be seen in Appendix A.

A create device message:

```
{
    "r": "hvny6",
    "o": "websocket",
    "dv": {
        "id": "websocket"
    }
}
```

Update a temperature range on a thermostat virtual device:

```
{
    "dv": {
        "css": {
            "temperatureMin": "18c",
            "temperatureMax": "22c"
        }
    },
    "r": "ted13",
    "o": "thermostat",
    "s": "#thermostat"
}
```

A message forwarded from the hub to a specific device:

```
{
    "r": "t7mv4",
    "o": "hub",
    "s": "#WEB312",
    "t": "update",
    "st": 1
}
```

FIGURE 5.13: Example messages.

5.2.2 Discussion

5.2.2.1 Possibility

This experiment was principally one of possibility. It took a concept from Experiment 1 and adapted it, grew it and implemented it into a working system. While the system was far from production ready, it remained largely stable for several months. It demonstrated that it was possible to describe a somewhat simple environment as shown in Figure 5.14, using the DOM. It also showed that the state could be held in CSS and that CSS selectors were a valid method for selecting devices and routing messages to them. It not only showed that JavaScript was a useful language for controlling the environment but also, from the evident, albeit limited success with Arduino, that the language used is less relevant than the hierarchical structure of the digital twins.

The key aims of the experiment were all realised: The WoT system was fully functional; it used the DOM, CSS and JavaScript; and showed that the theory could be applied to a real environment. Actions were converted to messages by the clients and those messages passed through the system to their correct destinations. Devices were built which had IDs and classes assigned to them and the environment was 'styled' like a Web page.

```
<context>
    <device></device>
</context>
```

FIGURE 5.14: The IoT environment, represented using the DOM.

This systems possibility meant that it comfortably satisfied the first aim of this thesis, namely to represent IoT devices within a DOM. While Experiment 1 theorised that this would be possible, the system in this experiment showed that it is. While it had many failings from a practical point of view, it nonetheless met the most basic aim.

It also came close to meeting the second aim of this thesis which is to build an IoT control system using only browser technology. While it only used the DOM, CSS and JavaScript, it did so in a way that was not an entirely standard approach. The DOM is hidden on the hub and not available to developers and it uses non-standard and somewhat unfriendly messaging approaches.

5.2.2.2 Practicality

Initially, there was not too much consideration for the practicality of the system or its approach, and important features of a production system, such as latency, were ignored in favour of demonstrating its feasibility.

From a developmental perspective, the processes for building either a new physical or virtual device were relatively similar and straightforward. A physical device could be any existing device with an API that could be integrated with the ub.js library. Standalone virtual devices were implemented similarly, meanwhile plugin-type devices were written as a simple JavaScript function or as an instance of a class that would push messages directly to the internal 'bus' channel.

An important advance over Experiment 1 was the reduction in duplication. Each CSS rule or command could be applied to many elements, simply by assigning all the elements the same class. This means that any number of elements would have the same state defined by a small set of rules. This is simpler to read for the developer, and there is less chance for an error to occur here. Of course, errors can still be introduced by an incorrect class applied to an element or an incorrect value in a CSS rule. However, unlike the previous implementation, where an error could be small and go unnoticed (e.g. an incorrect colour on one element), these errors are much more apparent. An incorrect class would be self-evident because the element would have no state at all as well as an incorrect value as all elements would have the same error visible.

The new approach is also more efficient for scripting. Rather than using JavaScript to loop over all elements to read or write their state, a class or property can be changed and the developer can rely on the built-in browser functions to carry out the update, a process that has been greatly optimised by browser maintainers and which is implemented in more efficient languages than JavaScript. This preference for built-in functionality is an important part of this thesis and one which is pushed further in Experiment 3.

An unintended benefit of using CSS allowed for swapping entire style sheets at once to change the state of an environment, like themes on a Web page. This is discussed further in Chapter 7.

However, this approach is not without its downsides. An obscure but important issue was the lack of CSS Custom Property support in the Node.js packages used. Custom Properties [222] allow the developer to specify their own property names by simply prefixing the name with '`--`'. This absence meant that CSS properties had to be parsed manually and acted upon using custom logic. While this was necessary, it was contrary to the aim of using built-in browser technologies. This required the use of an allow list of CSS properties that could be found and used by the hub application, although this meant that the final system was very closed and proprietary. It necessitated a rigid taxonomy rather than allowing a looser 'folksonomy' as may be expected in an unregulated system. Browser prefixes and varied browser API support are good parallels to draw here as they show companies implementing their own allow lists. The W3C releases specifications which are implemented by the main browsers, but despite this, they each develop their own features. In CSS these can be seen in the form of prefixes such as `-moz-` for Firefox and `-o-` for Opera. These are features that are either found in an unfinalised state at the W3C, or else are the browser's own proposals which may, or may not be progressed. While this system is a good illustration of how an individual browser handles a finite list of CSS properties, an IoT system would have to handle properties from many different client devices built by many manufacturers and it would not be possible to maintain a list of allowed prefixed properties for each. More modern virtual browsers (and all modern desktop and mobile browsers) implement the CSS Custom Properties specification to at least a degree which could make this form of manual processing obsolete. This concept is further explored in Experiment 3.

The reliance on a centralised hub also brings some difficulties in that it is a single point of failure within the system. If a hub fails, then all devices connected to it will be uncontrollable and the user will not be able to retrieve their state. A mesh or broadcast approach would be a potential solution to this, but it would add a lot of complexity to register devices with multiple hubs for redundancy and then ensure commands are not duplicated. This was partially investigated when exploring communications arising between multiple hubs, wherein a hub would forward all messages to other hubs, which would then act upon them, if relevant. This is illustrated in Figure 5.15. The system assigned a unique ID to each message so that this could be possible, although the increase in traffic would be non-linear.

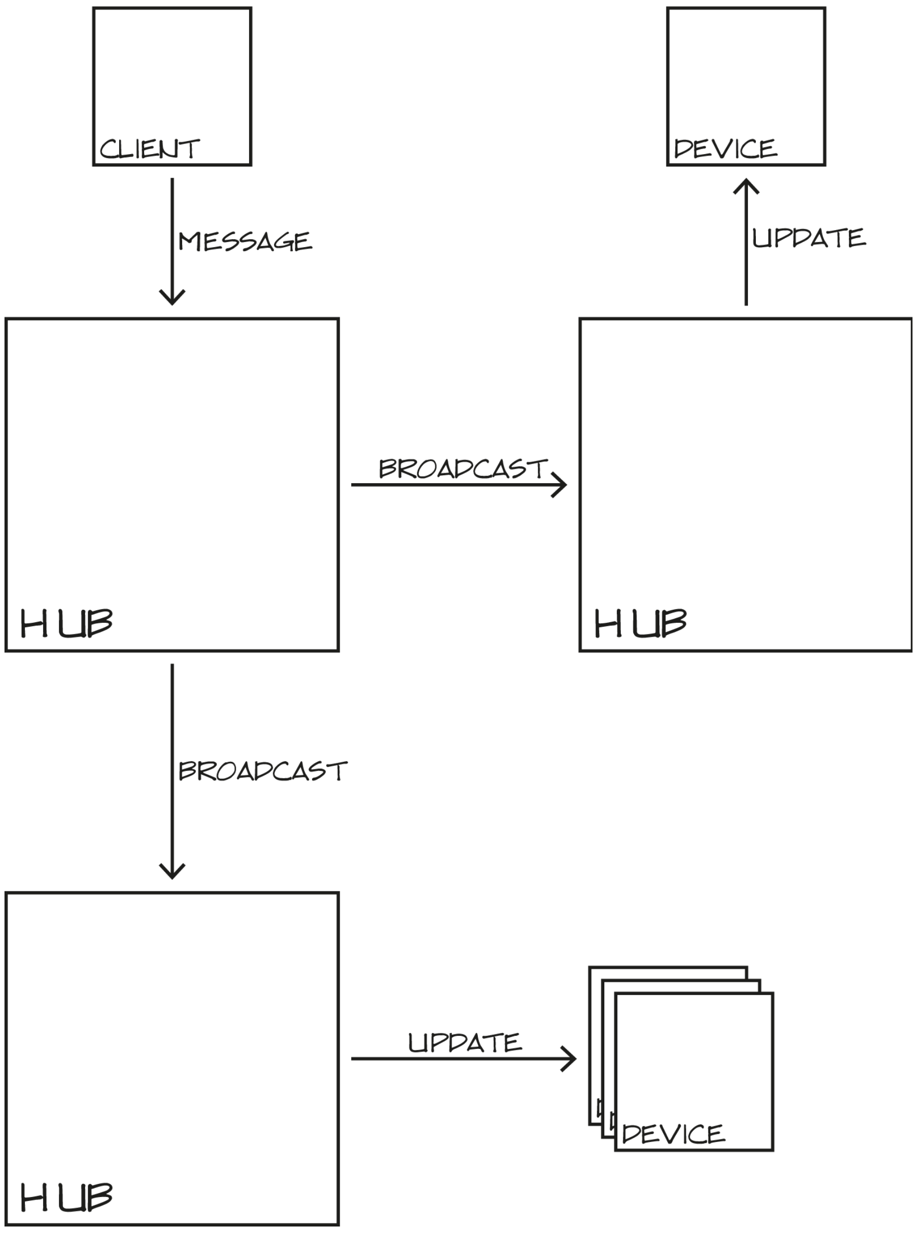


FIGURE 5.15: A hub forwarding messages to other hubs.

Latency in the System

A more practical, but less than directly relevant issue was that the latency within the system was quite high. This was partially due to the implementation, in that JavaScript is not a particularly fast language and may be held up by garbage collection which is automatic and impossible to schedule. Largely, however, it was a result of the number of hops necessary to control a device. A message from the user would have to travel to the hub, which would then query the DOM and then send it on to the relevant devices. These would then send a response back to the hub which forwarded it to the user. This flow of messages was in addition to any usual network hops. This journey was made worse by the hardware involved, as it included devices such as low-cost ESP8266 Wi-Fi chips and Arduinos connected over serial. This round trip time was often very obvious to the user and created a disconnect between the user's interaction and the subsequent action occurring in the environment, as triggered by the hub. It has been found that most users can detect a latency of over 40ms in a virtual touch-based system [223], however, it should be noted that similar research has not yet been conducted for IoT systems. Violating this threshold without UI mitigations, such as loading graphics, will make the user second guess their inputs and disrupt both their enjoyment and the flow of their interaction. This could have been improved with the caching of DOM searches, but the other factors were unavoidable.

Optional and Plural Responses

Another issue came about in trying to mirror HTTP responses in the architecture. A device that has matched a selector was free to return a response to inform the initiator that the action had occurred although, as this was optional, the initiator would not know if a lack of response was due to the device not existing, its failure to complete the action, or simply because it had not implemented responses.

The opposite problem occurred when multiple devices received messages. The initiator would receive multiple responses, but would not necessarily know how many it was expecting due to the zero-knowledge nature of the clients.

Possible solutions are presented in Chapter 7.

From a more practical standpoint, due to the system being experimental rather than production ready, it had a number of smaller issues that were never resolved. None of these however affected the outcome of the experiment, although they were recorded as areas for future improvement. These were:

- Due to the large amount of proprietary handling of CSS properties, the CSS cascade was not fully implemented. CSS has a highly defined hierarchy of overrides based on the specificity of a selector and the order in which the rules are parsed. This hierarchy was was too complex to re-implement for the purposes of this experiment. This, in combination with a desire to test the approach with a broader set of users, were driving forces behind Experiments 3 and 4.

- There was no conflict resolution, or indeed any concept of a hierarchy of importance or order for the messages; they were were acted upon in the order in which they arrived. There was the potential for a message with a simple selector to be sent after one with a complex selector but, due to the computation time, the complex message could have arrived at the device at a later point in time. This was never observed in practice, as most input was either manual or derived from sensors polled every few seconds, but it was also not accounted for in the engineering of the system.

- The internal publish/subscribe model of the 'bus' object was inefficient and insecure. There were no restrictions on reading messages and the large number of subscribers using event triggers could very well have accounted for a great deal of the latency observed within the system.

- The communication used by the system was based on JSON, but did not follow any particular standard schema and so, therefore, was adding a proprietary technology to an already fairly overloaded space. It contained abbreviations and single-character properties as the devices using it did not have sufficient memory to store long variable names and parse long JSON objects. This would have made it hard for developers to understand.

- The hub relied on PhantomJS to hold the DOM and, while this is a full-featured WebKit browser, development was suspended in 2018 [214] and so has only had minor updates since. It also never reached the level of development that desktop browsers have.

- The hardware that controlled the radio frequency plug sockets was never very reliable and so it could not be guaranteed that a device was in the same state as the hub's DOM said it was. This manifested on at least two occasions during testing where the radio-controlled plug socket repeatedly failed to receive messages due to interference or an internal error and the safe, oil-filled heater did not turn off when it was unaccompanied. A workaround was to send each message five times with a one-second spacing. This issue was amplified because the plug sockets could not send success or failure messages back.

Overall, the system built was a very practical research system, but not sufficiently practical for the wider community, a conclusion which helped to define Experiment 3 which is aligned even more closely to extant Web standards.

5.2.2.3 User Acceptance

From a research point of view, the system was more than adequate for demonstrating that a WoT system could be built using the DOM as a foundation, especially as there was only one developer involved. However, it was not suitable, even for testing, within the wider community. The system was overly complex and took too long to develop hardware and software for it to be useful outside of the prototype system. Any tests of user acceptance were not possible, as there was no way to allow developers to interact with it without requiring several hours for them to become properly familiarised. This, and technological advances, serve as drivers for Experiment 3. In this respect, it fails the third aim of this thesis, which was to produce a system acceptable to developers.

However, it is still possible to assess it through a heuristic analysis.

Heuristic Analysis

1) Visibility of system status The status of this system was almost deliberately hidden within the hub. As such, it was somewhat of a black box to any device or client connected to it. Visibility was achieved only through the Web interface or else by directly looking at the device in question. For an end-user, this would be fine although, for a developer, it is less than ideal. A developer could not know the contents of the DOM easily unless they took a copy from a specially made page on the Web UI and this was a snapshot that did not reflect changes in real time.

Messages in the system could be monitored as they were broadcast. However, the format was not friendly to humans, as it employed a lot of abbreviations to save space on constrained devices. A converter could have been written to parse it into a longer form, but it would still have been a non-standard schema.

2) Match between system and the real world The system matched the real world by reflecting changes arising between the device and the digital twin on the hub in that any change to one was mirrored to the other. This mirroring helped the developer and user to interpret the state of the environment, as every state in the hub had a real world equivalent, meaning that metaphors were largely unnecessary. However, the Web UI used metaphors for sliders and switches with its virtual devices. These virtual representations of physical interfaces were in line with the users' expectations of control (e.g. a switch for a lamp). This type of interaction should have been both familiar and intuitive. However, this was limited by the creative talents of the developer of the system.

3) User control and freedom Developers had a lot of freedom to create devices, both physical and virtual, and then connect them to the system. The system itself had the facility for plugins and modules within the hub, but the interface to this was proprietary and, as of writing, is already obsolete technologically speaking.

4) Consistency and standards As mentioned, the system was quite proprietary in its architecture, despite using a lot of open standards. It used an internal event-based messaging system which was not standard, and the messaging format, while JSON, was developed specifically for this project. This is an easy trap to fall into and one which is not unique to this system.

5) Error prevention The system was incomplete with respect to error prevention. In the case of a spelling mistake, it could have silently sent messages to the wrong device which could have had negative consequences or, at the very least, left the developer or user confused. The lack of a proper response framework hindered this further.

The system also required a lot of knowledge of how it worked and why, before anything could be written for it, which presented a very considerable barrier to enabling Web developers to transition to a WoT system of this type.

6) Recognition rather than recall The jQuery-like client was the most usable part of the system as it was deliberately written to be familiar to the developer. However, as of the time of writing, jQuery is losing popularity and, very soon, will become obsolete. A better alternative would be to use proper Web APIs for selection, but with this architecture, it would be a complex proposition to develop and it could potentially create instabilities.

The hub code was written in JavaScript and so, as with any application, a skilled developer could familiarise themself with it, although they would need to learn its idiosyncrasies. The data structures would be familiar, but the overall architecture was as unique as any unframeworked application. If a developer specialises in front-end Web development, then they would probably have trouble developing for the hub or, if they specialise in back-end development, they may have trouble developing for the clients. Due to the lack of transparency in the hub, potential developers require a great deal of knowledge before creating a hub plugin, but much less so to connect a client.

7) Flexibility and efficiency of use Flexibility was an important part of the system and the range of client libraries – Node.js, browser, embedded JavaScript – allowed developers to add various devices and components, connect to external services, and build virtual devices fairly easily. They were also free to layer frameworks on top of the libraries to make their work faster. However, the client libraries were nowhere near as compatible with external frameworks as the system used in Experiment 3.

8) Aesthetic and minimalist design The system was minimalist in what it exposed, in that it only exposed a messaging interface with a limited command set. Developers could send messages to the hub which then forwarded them to clients and they did not necessarily need to know what happened inside the hub to do so. However, this came at the expense of information, particularly in relation to errors.

9) Help users recognize, diagnose, and recover from errors The system did not report errors effectively. Messages would either have a target in the DOM and be forwarded on, or they would not and would therefore stop at the hub. A comprehensive error reporting system, similar to HTTP response codes, could be produced and would help with error diagnosis. However, in the case of multiple hubs, it may be impossible or inefficient for any single hub to know that there are no matching targets and so it may not be possible to send an equivalent of a 404 response.

10) Help and documentation A long document was written as a guide to the ub.js system. However, the fact that a long and fairly confusing guide was required is an indicator that the system was not developer friendly. Many concepts required explaining and the client similarity with jQuery could have easily brought confusion as developers would have had to remember or look up the differences between the two.

5.3 Experiment 3: A Distributed, Browser-Based, Web Components WoT System

This experiment built on Experiment 2 in a few important ways. First, it was more closely aligned with Web standards, largely thanks to advancing technology. Second, it aimed for simplicity, and third, to be a system that could be tested within the wider community. It retained many of the core approaches of the previous experiments, using the DOM, CSS and JavaScript for similar purposes.

The limited scope of this approach blurred the lines between an IoT control system and digital twins, as the devices were assumed to exist but not actually implemented. The digital twins were acted upon directly rather than via an interface to a hub. Devices and their components were again arranged within a DOM tree and the same hierarchical approach was taken to modelling them. However, in this implementation, they took advantage of more Web standards than before, as several standards had been finalised and integrated into the core Web browsers since the last experiment took place. This experiment was more superficial than the previous one because it only simulated an environment and devices, rather than actually controlling them. There was no back end and commands were not sent outside of the browser. It was still a fairly complete and very flexible WoT management system and, in a real situation, commands would have been forwarded from the DOM element digital twins to the physical devices. It was assumed that this connection to real devices is both possible and irrelevant to this experiment as per the findings of Experiment 2.

5.3.1 Technical Implementation

The system handled the representation of devices, their state and their affordances using Web technologies while, at the same time, assuming there would be either a local or remote service that detected devices, registered them and proxied control messages to the correct format. Given that the second approach in the previous section demonstrated that control of devices using CSS is both possible and relatively trivial, this implementation focused on the developer experience and the use of more standardised Web APIs. Here, Web Components were used to represent devices, and CSS Custom Properties were used to control and record state.

The key difference between this approach and the previous one is that the system ran entirely within the browser. The DOM was held locally within the user's browser and would thus be duplicated across all instances of the system. Multiple users could have opened multiple copies of the Web page and each would have been shown a representation of the same environment. A hub with a controller was no longer necessary, as all the control and state management happened within the Web page instance. Changes to the environment

DOM would have been pushed to other instances and the appropriate devices. The complete system may still have needed a hub, but it would have been limited to a bridge connecting the instances and devices. The hub may have had another instance of the DOM, but would not necessarily have needed to have any primacy over the browser instances, unlike the hub delineated in Experiment 2. The architecture of this system can be seen in Figure 5.16.

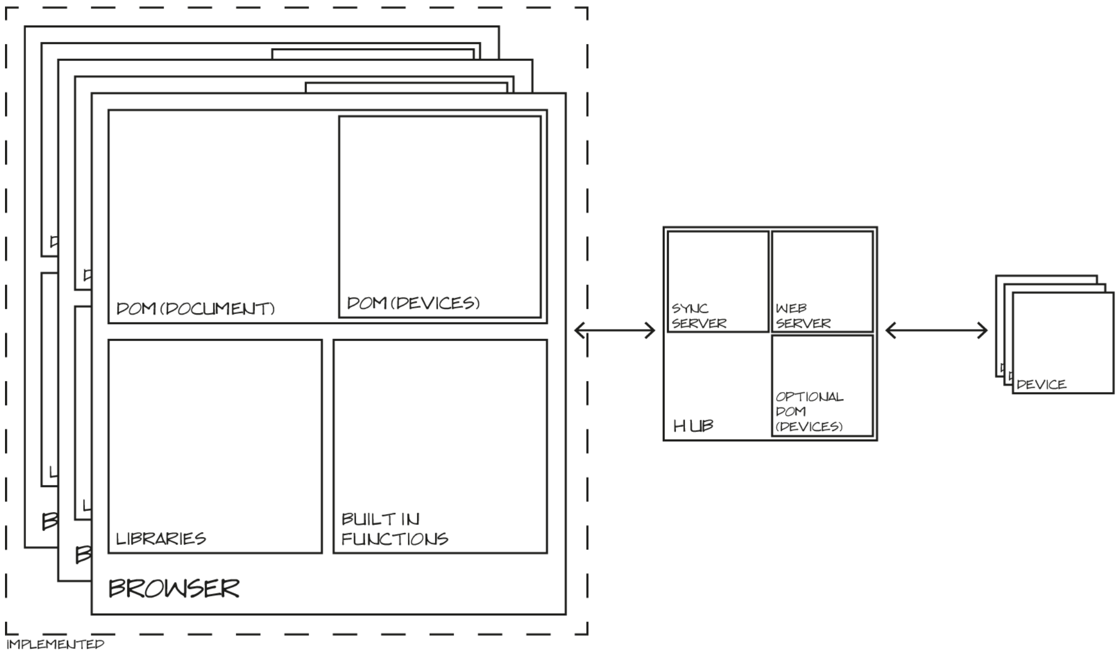


FIGURE 5.16: The architecture of a complete system, with the implemented part indicated.

The DOM in the browser contained digital twins for all devices and components of devices within the environment. Each device and component was modelled as an HTML element in the DOM using the Web Components. The instance of each Web Component was placed into the DOM tree either at the appropriate place, or multiple places if there were multiple instances of similar devices. An example DOM can be seen in Figure 5.17. Shifting much of the logic to the browser meant that a server in this system would have simply been a router sending commands between browsers and devices, mitigating conflicts, holding composed devices, and potentially caching their respective states to prevent flooding the IoT devices with requests. The logic the server could employ here is far less interesting from the point of view of the research in this thesis, as it has already been solved for so many Web sites before and thus was ignored in this implementation. Addressing each function of a potential hub: the process of routing commands is very trivial; conflict mitigation is another area of research entirely; composed devices have been demonstrated in the previous implementation; and while caching can be implemented with varying levels of complexity, for the current purposes it is almost inconsequential. Synchronisation and hierarchy of messaging importance would be an interesting avenue, although tangential to the core aims of this thesis. For simplicity's sake, it was assumed that there was a single user of the system and that all of their commands held equal weight (i.e. a command to flash a light for an incoming message was treated equally to one to flash a light for a fire alarm).

```
<html>
    <body>
       <iot-room>
        <iot-door />
        <iot-cooker>
            <iot-hob>
                <iot-hob-ring id="hob_ring_859" />
                <iot-hob-ring id="hob_ring_194" class="power-on" />
                <iot-hob-ring id="hob_ring_504" class="power-on" />
                <iot-hob-ring id="hob_ring_439" class="power-on" />
            </iot-hob>
            <iot-clock />
            <iot-switch />
            <iot-oven id="oven_958" />
        </iot-cooker>
        <iot-table />
        <iot-fridge-freezer>
            <iot-fridge class="iot-cupboard open too-hot" />
            <iot-freezer class="iot-cupboard" />
        </iot-fridge-freezer>
        <iot-counter />
    </iot-room>
    </body>
</html>
```

FIGURE 5.17: An example DOM using Web Components showing a basic kitchen.

CSS Custom Properties were used to overcome the allow list limitation of Experiment 2 and enabled arbitrary state to be controlled with CSS and stored within CSS rules. The `power-on` class could contain a property called `--power` with a value of `on`, as well as any other standard CSS for a document. This property could be read by the element representing a device and, if appropriate, the element could act on it. A state diagram of how this process occurred can be seen in Figure 5.18.

Working Around Browser Limitations

The step where an element acts on a change to its style is the only time a workaround had to be used to implement this approach. HTML elements are not inherently aware of changes to their style, as it does not affect the behaviour of built-in elements. The browser is aware of the style change and applies presentational changes, but the underlying custom element is not notified that its appearance has changed. To work around this, a `window.getComputedStyle()` call was added when the style or class of an element was changed. However, this is imperfect at best, and broke down in some cases of inheritance, for example when setting a class on a parent would trigger the children to re-render but would not trigger any actions defined in the Web Component.

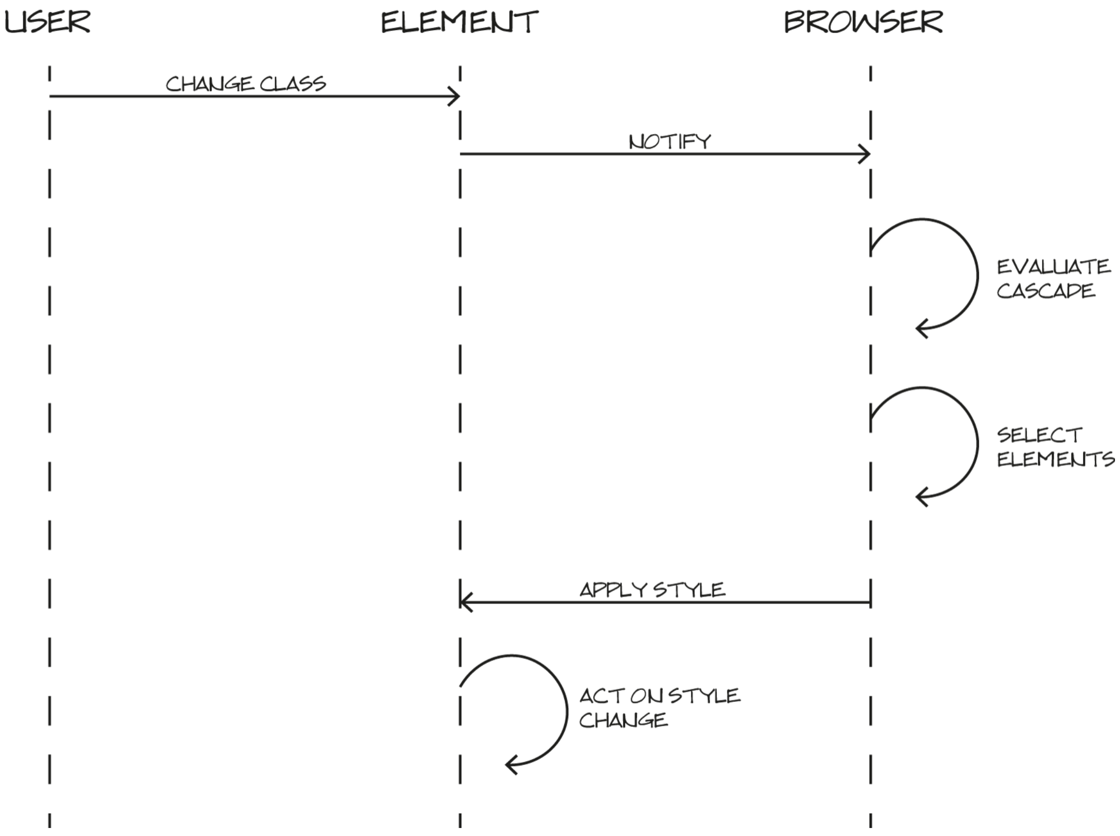


FIGURE 5.18: A simplified sequence diagram showing a custom CSS property changing as a result of a class change.

The use of CSS Custom Properties allowed for using the browser's built-in interpretation of cascading rules in CSS, which is a marked improvement over the previous implementation, and a milestone in reaching full integration with Web standards.

The architecture of the Web Component was designed to align with the principles of the W3C's WoT implementation. Each device class is inherited from a generic `WoTElement` class, which itself is inherited from the `HTMLElement` class. This reliance on the HTMLElement parent was a requirement of the current browser implementations of Web Components and could be removed in a later version. The WoTElement contained `properties`, `actions` and `events` which are similar to, but slightly limited versions of, the W3C's WoT definitions. It also utilised the Shadow DOM for presentational markup which generated a visual representation of the device. This visual representation was a 3D version of the device within a 3D environment, but it could also be a UI element within a control interface. This allowed either the control UI to be separate from the device DOM or for the device DOM to generate the interface from itself. This was briefly experimented with by making the elements of the 3D representation clickable, wherein a click turned them either on or off, or closed or opened a door. This was removed from the user study in Experiment 4, as it was not relevant to that specific line of investigation, but its potential is discussed in Chapter 7.

Properties, Actions and Events

The W3C's WoT Group defines these as [87]:

Property: an "affordance that exposes the state of the thing", which can be read from and written to.

Action: an "affordance that allows to invoke a function of the thing".

Event: an "affordance that describes an event source".

The interpretation of these within this specific implementation was loose, largely due to the evolving nature of the W3C's document. In this system, the following definitions were used:

Property: the state of a thing, which can be read or written.

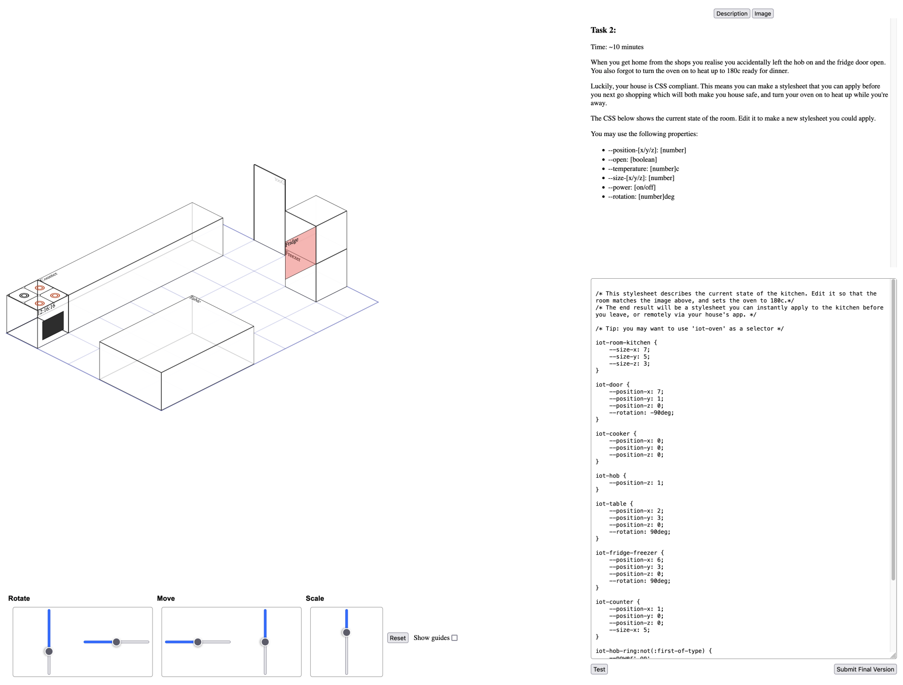
Action: a function that can be invoked on the thing.

Event: an event that is initiated by the thing, and which can be subscribed to using an event listener.

The result of using the DOM in this way meant that it was directly available to the developer, as they could use the browser's built-in inspector tool to view and edit the DOM itself. This was a great step forward from Experiment 2 which hid the DOM within the hub. This is not only a benefit for visibility, but also allows developers to follow a more familiar debugging experience, or even potentially use browser plugins to aid them.

5.3.1.1 The User Interface

Unlike Experiment 2, the UI for the system was designed for developers and made to fulfil the requirements of Experiment 4 for which it was later used, and therefore it was not conducive to use by an inexperienced user. There was an interactive 3D isometric representation of the environment built from the devices within the DOM themselves. This preview could be panned, zoomed and rotated to allow the developer to see the effects of their changes on the environment. This, too, is a marked improvement on the previous experimental system. On the right were a section for instructions and a text box containing the current CSS rules for the environment. Editing the CSS rules and pressing the 'Test' button applied the rules to the Web page's DOM. As the devices were a part of the page, there was no separation between style that was applied to the page and the style applied to the devices. However, on this page there was a cosmetic function that prefixed any rules with a namespace so they would only apply to the devices. This was a user experience feature to prevent participants from inadvertently changing the document and breaking the experimental interface. The layout can be seen in Figure 5.19.



Task 2:

Time: ~10 minutes

When you get home from the shops you realise you accidentally left the hob on and the fridge door open. You also forgot to turn the oven on to heat up to 180c ready for dinner.

Luckily, your house is CSS compliant. This means you can make a stylesheet that you can apply before you next go shopping which will both make you house safe, and turn your oven on to heat up while you're away.

The CSS below shows the current state of the room. Edit it to make a new stylesheet you could apply.

You may use the following properties:

- --position-[x/y/z]: [number]
- --open: [boolean]
- --temperature: [number]c
- --size-[x/y/z]: [number]
- --power: [on/off]
- --rotation: [number]deg

```css
/* This stylesheet describes the current state of the kitchen. Edit it so that the
room matches the image above, and sets the oven to 180c.*/
/* The end result will be a stylesheet you can instantly apply to the kitchen before
you leave, or remotely via your house's app. */

/* Tip: you may want to use 'iot-oven' as a selector */

iot-room-kitchen {
    --size-x: 7;
    --size-y: 5;
    --size-z: 3;
}

iot-door {
    --position-x: 7;
    --position-y: 1;
    --position-z: 0;
    --rotation: -90deg;
}

iot-cooker {
    --position-x: 0;
    --position-y: 0;
    --position-z: 0;
}

iot-hob {
    --position-z: 1;
}

iot-table {
    --position-x: 2;
    --position-y: 3;
    --position-z: 0;
    --rotation: 90deg;
}

iot-fridge-freezer {
    --position-x: 6;
    --position-y: 3;
    --position-z: 0;
    --rotation: 90deg;
}

iot-counter {
    --position-x: 1;
    --position-y: 0;
    --position-z: 0;
    --size-x: 5;
}

iot-hob-ring:not(:first-of-type) {
```

FIGURE 5.19: The user interface of Experiments 3 and 4.

5.3.2 Discussion

5.3.2.1 Possibility

Much of the possibility of representing an environment using the DOM was settled in Experiments 1 and 2. However, this experiment took advantage of some new technologies to bring this closer. The use of Web Components and Custom Properties allowed for standards-based digital twins to be produced in a very lean and efficient way. The total size of the `WoTElement` class was less than 150 lines of code, including comments and white space, as compared to the many hundreds required to achieve a similar outcome in Experiment 2.

This experiment also showed that the approach chosen could be aligned with the W3Cs TD approach and that the two could work together harmoniously. While it was not explicitly tested, it would be reasonable to assume that if the two were aligned completely, a class of device within the DOM could be automatically generated from a TD.

5.3.2.2 Practicality

The approach was much more practical than Experiment 2, as indicated by the simplicity of the code required to produce it. Any developer new to a system like this would have far less to learn than one using the system in Experiment 2. Despite its relative simplicity, the system still demonstrated an effective and fast method of controlling IoT devices via their digital twins using only Web technologies. It also did this in a way that followed not only the technologies, but also the general approach to making a Web page. This was a significant contribution to both the second and third aims of this thesis, as it followed Web standards and had a relatively low barrier to entry for Web developers. The latter was tested explicitly in Experiment 4.

However, some limitations could be addressed in future. First, browsers do not notify an element's JavaScript representation that its style has changed. This presented a problem that could not be adequately overcome within this experimental system. However, the browsers clearly have an internal event that triggers a re-render and so it should be possible to expose this event to the JavaScript engine. This would require a custom build of the browser that would technically break the requirement that the system be built within the constraints of existing browsers and thus was beyond the scope of this experiment.

Second, the use of CSS for storage of state was shown to be less than ideal. It worked very well for defining rules to apply state to elements, and Experiment 2 showed it was a good method for routing messages through the system. However, the use of a style sheet to store state makes reading it very inefficient. This is because the browser must evaluate the whole cascade to return a value for a property every time it is read. This is not dissimilar from one of the issues with Experiment 1, wherein the inheritance of properties would have required the browser to traverse the DOM tree. While the values for properties are almost certainly cached, making the read action an O(1) operation at the time of execution, it is still not ideal as this value has to be calculated at some point.

Additionally, CSS was intended to store presentational state and not data, and so is limited in many ways. For example, it has no concept of permissions or security and so cannot police whether an action should be allowed. It also does not have properties which can only be read from and not written to (e.g. the temperature from a sensor can be read, but writing to it will have no effect). An alternative may be to have a backing database of state, possibly using the Indexed DB [224] specification within the browser, and to employ CSS to request changes to the state rather than to contain both the changes and state. This would naturally add complexity and would require more work (as outlined in Chapter 7) but would remain within the limitations of the browser.

5.3.2.3 User Acceptance

User acceptance of this approach will be assessed in far more detail in Experiment 4 in the next section, although a heuristic analysis is still useful to explore some of the underlying approaches rather than the interface.

Heuristic Analysis

1) Visibility of system status This system had excellent visibility, with the caveat that all the changes were occurring locally and not replicated to a server or devices. With each digital twin in the environment under control having a single wireframe representation on screen, it was clear how they related to one another and what the effects of commands were. This approach to displaying an environment should work for any single room or group of devices, although it would need to be reconsidered for more complex deployments (e.g. the very large concert environment in task 4 of Experiment 4 was hard to navigate and too detailed to see all at once). Due to the inherent flexibility of the system, any environment could be made to produce any number of representations and it could present these simultaneously to the same user or many users. Making this experience user or developer friendly would ultimately be the job of user experience and graphical designers.

2) Match between system and the real world The preview was arranged to be visually similar to a real room, with the key objects and furniture represented in some way. This 1:1 mapping of representation to a device was a product of this heuristic and as such it provided a reasonable facsimile. With a better 3D engine and 3D modelling skills it could have been vastly improved, but may then have required an interface from the 3D scene to the DOM which would have added further complexity. The UI was a compromise to make a usable system in a reasonable time and this is very apparent in places.

Conceptually, the space in the UI responded similarly to the real world scenario, having three dimensions. However, it lacked many physical properties, notably gravity and collisions between objects. Within the representation, many devices could occupy the same space and they could be floating above the ground without support. The potential work needed to overcome this is discussed further in Chapter 7.

In terms of status, the system did an excellent job in communicating this, both through the CSS and via direct access to the DOM, and also by allowing the user to see at a glance the colour of a light or the position of a chair.

3) User control and freedom With the adoption of Web standards, the expert user gained unprecedented control over the system and, therefore, the environment to which is was connected. Within the confines of the UI, they could change the CSS to alter the environment, but beyond this they could potentially write JavaScript or browser plugins, or even export the DOM to an external system.

During the production of this experiment, it became clear that it was trivial to use an IDE 'off-the-shelf' without modification to create and control the interface for the environment. This is very important as it is a cornerstone of the aim to make the WoT widely available to Web developers. With an IDE comes complex editing, syntax highlighting, source control integration, linting, and hundreds of other tools and integrations. Along with this complexity comes advanced debugging and error handling from tools that already exist. Very few other IoT or WoT systems can offer this.

4) Consistency and standards The system conformed to Web standards in all but one place which means that, in theory, any Web developer could be given access to the source code and be able to use it within a very short period. Experiment 4 explored this in greater depth by allowing real developers without knowledge of the system to use it.

5) Error prevention Error prevention arises in two instances, firstly during the creation of the code, and secondly in the use of it. The latter is entirely in the hands of the developer. In this system, there was very little error handling, as it was assumed that the developers using it would be sufficiently knowledgeable to produce error-free code. It was also too large an undertaking to produce error correction and handling for the UI. There was, however, full access to the browsers' developer tools from which users could track down errors for themselves.

One major benefit of using standard approaches is that, during the creation of code for the system, the developer's IDE would provide linting and syntax highlighting. This would reveal many syntactical errors in familiar places and logical errors would appear as normal in the browser's console at run time.

6) Recognition rather than recall This approach excels at recognition as every part of it will be familiar to a Web developer. With the addition of Web Components and Custom Properties, the journey to a Web-based solution is almost complete. Web developers would recognise the format of the CSS and JavaScript, as well as the DOM behind an environment. This was also explored further in Experiment 4.

7) Flexibility and efficiency of use The openness of the approach meant that there was almost infinite flexibility in approaching and solving a problem. However, the structured and well-known boundaries of Web technologies mean that transition from Web programming to WoT programming could be achieved very efficiently.

The ability to create a UI meant that the end-user could be shown a very simple or very complex interface, or one which can be tailored to their skill level. The developer, too, had the option to use simple Web APIs to achieve simple goals or layer pre-existing and well-known libraries and plugins over the top of the DOM to produce vast and complex solutions.

The programmatic efficiency of the system was based on over 30 years of advancements in Web and browser technology, and so began its life with all of the pre-existing shortcuts and optimisations. This facet alone gave it a vast benefit over almost any other non-Web-based IoT solution.

8) Aesthetic and minimalist design Interaction design is not a key driver of this experiment, although it is an important measure of a well designed system, as one that is not well designed has limited options for interactivity. Take, for instance, the Arduino. When it was first released it was an incredibly flexible platform in terms of features, but had to be interacted with through a proprietary IDE containing many bugs. It was easier to use than any other embedded platform and many people persisted with it. However, it suffered because the developers had to do a lot of work to achieve parity with the IDEs for other areas of development. The WoT could easily inherit this problem, but this approach does not. With it, developers are free to take their current IDE and program the WoT almost instantly.

In terms of the end-user, the design of the UI in this experiment, while deliberately minimalist, demonstrated how straightforward it could be to create a UI for a WoT deployment. However, even that interface could have been reduced to text, speech, or braille, if the developer so chose. In fact, due to Web integration, it could offer all three accessible options with very little change to the underlying code. This prospect is discussed further in Chapter 7.

9) Help users recognize, diagnose, and recover from errors Web developers have a shared domain language of error codes and syntax, one which is acquired over time. It is baffling to the novice yet routine to the seasoned expert. While this system did not offer errors in plain English, it presented them in such a way that would be familiar to an expert Web developer. They appeared in the same console and were written in the same syntax. They could be thrown in the same way in the JavaScript code as for any other Web-based system and they could be investigated using the same tools. While learning how to understand errors such as these is not trivial, it is transferable back and forth between this approach and normal Web development and so the effort is not wasted nor expended multiple times.

10) Help and documentation The documentation for the system is almost entirely the same as that for the Web. It has been written over the last 30 years by teams of experts addressing the underlying technologies and by those who specialise in accessible technical writing. There are third-party tutorials pertaining to every aspect of the construction of the system used in this experiment and nothing is proprietary. In this way, to an expert Web developer, the system was self documenting. However, the application itself did benefit from some additional explanation in the form of documented code. Due to the distributed nature of the knowledge behind the approach, no specific help or guidance was needed, although it would most likely be appreciated if the approach were to be applied on a wider scale.

5.4 Experiment 4: Assessing Developer Acceptability of a WoT System: Designing CSS for an Environment

In this experiment, participants were asked to complete a series of tasks using the system developed in Experiment 3. These were designed to test whether the approach used is intuitive and can be learned swiftly. An expert sampling approach was used, and the participants chosen were all software developers with some experience in Web development. They were recruited from personal and professional contacts as well as through targeted posts on Reddit and LinkedIn. The study was completed using the interface and system developed in Experiment 3 and a data collection back end written in JavaScript.

Participant Sources

LinkedIn

The author's LinkedIn network is composed of distinct groups of people:

- Those in and around London, as that is the largest city near to where the author grew up, went to university in, and had their first job.
- Students and staff from the Electronics and Computer Science department at the University of Southampton.
- Professional and personal connections from Australia and New Zealand.

Reddit

A Reddit account was created for this project, and was publicised in the following subreddits:

- /r/css_irl - for people who take photos of real things they see that look like a developer has made a mistake with their CSS syntax.
- /r/homeautomation - for discussion of home-based automation and IoT deployments.
- /r/webdev - a group of professional and amateur Web developers.
- /r/IOT - discussion about the IoT, both from an industry and practical perspective.
- /r/css - a group of enthusiasts who share developments in the CSS specification, as well as uses of it and those seeking help.

The study had three aims:

1. Discover whether an expert developer can learn the approach promptly
2. Begin to understand how developers view an environment, and whether that aligns with a hierarchical representation
3. Assess the acceptability of the approach to expert developers

Each of these aims was assessed differently, albeit within the same study.

5.4.1 Participant Breakdown

All the results shown here should be framed by the abandonment rate. Of 61 original participants, only 22 completed the study, with a further two offering feedback at a later stage. Some 25 completed all four main tasks and 22 submitted the final scene and feedback. Two additional pieces of feedback were gained through a request for feedback from those who did not complete the study. This was possibly due to a higher abandonment rate than would have been realised had the study been carried out in person, as the social pressure from walking out of a room mid-experiment is higher than closing a browser window. However, 22 is a sufficiently large cohort to draw some useful conclusions [208]. As can be seen in Table 5.3, there was a consistent loss of participants between tasks. This may suggest that they found the tasks hard, uninteresting or otherwise time-consuming, yet advanced to the next stage in the hope of some kind of improvement or that it may be the end of the ordeal. However, this is mere conjecture as data on abandonment was not collected.

Demographic data was requested as it colours the results, although more extensive demographics were deemed excessive and therefore not collected. Age was used to screen out those who were under the age granted by the ethics committee (18) and self-reported experience was collected as it may have been linked to acceptance or understanding of the technology.

As shown in Table 5.1 and Table 5.2, the participants were spread over a wide age range, but were skewed heavily towards younger ages for both populations. Despite the younger ages, all participants who completed the study had at least 1 year of prior experience in Web development and at least 2 years of general development experience, while some had much more than this. The participants who completed the study may have been slightly more experienced in Web development, and development overall, as none of the least experienced developers completed the study (although the median development and Web development experience remained the same for both groups). Overall, the shape of the population which began the experiment and those that finished was mostly the same, and so no conclusions can be drawn as to why some of them may have abandoned the study.

Age	
Mean	25.30
Median	24
Mode	18
Min	18
Max	52
Length of experience	
Median	5-10 years
Mode	2-5 years
Min	less than 1 year
Max	greater than 10 years
Length of Web-specific experience	
Median	2-5 years
Mode	5-10 years
Min	less than 1 year
Max	greater than 10 years

TABLE 5.1: Results from all 61 participants who began the study.

Age	
Mean	23.82
Median	23
Modes	18, 21, 22, 25
Min	18
Max	33
Length of experience	
Median	5-10 years
Mode	2-5 years
Min	2-5 years
Max	greater than 10 years
Length of Web-specific experience	
Median	2-5 years
Mode	2-5 years
Min	1-2 years
Max	greater than 10 years

TABLE 5.2: Results of the 22 participants that completed the entire study.

5.4.2 Study Structure

The study was arranged with the following provisions:

- An introduction page, with a brief description of the study, its purpose, and links to the ethics approval.
- A structured demographic and experience questionnaire.
- An image of a scene, for which the participant was asked to describe it without prompts for the preferred structure of the answer.
- Four tasks, each of which used the interface and system from Experiment 3.
- An image of the same scene as previously presented, but this time with a light negative prompt for the structure of the answer.
- A semi-structured questionnaire about the participant's opinions of the approach used by the study.
- A thank you page informing the participant that the study was complete.

5.4.3 Part 1: Learning the Approach

Their ability to learn the syntax was assessed by giving the participants four problems to solve. These problems began very simply and became progressively more complex. This learning approach should by now be familiar to any developer, as it is how most practical university work is completed, and the way in which most online tutorials for Web development are structured. Each step was accompanied by a list of CSS Custom Properties that could be used to solve the task, a brief, and an image of a scene. The user could interact with the scene using a text box which contained an initial incorrect or incomplete description of the scene using CSS, and they could then see the results in a panel showing a dynamic scene created from a provided DOM and the CSS in the text box. The interface can be seen in Figure 5.19 in Experiment 3. The participant then had to correct the CSS to make the dynamic scene match the image and brief. The exception was task 4 which only provided an open-ended brief. Results were saved for each attempt at each task and also for the final version.

The experiment was carried out remotely. This was partially as a result of COVID-19, but it also offered the opportunity to reach a wider variety of people without geographical limitations.

Participant Rewards

The participants were motivated with a chance to win a £50 Amazon voucher, with an additional voucher being added per 10 study completions. This was an attempt to encourage sharing as the participant would not suffer a diminished reward for encouraging others to take part. This method of encouragement was chosen because a single £5 reward did not seem sufficient to motivate professionals who would typically earn many times that per hour and the potential to win more may have been a bigger motivator, despite the final payout per person remaining the same.

5.4.3.1 Task 1

The first task was an object which needed to be moved in two dimensions and had the following prompt, followed by a list of available properties. The prompt provided was:

> In this first task you will reposition and close an IoT fridge, and set the correct temperature.
>
> Edit this CSS to set the temperature of the fridge to 4°C, and close it. Then change the room and fridge to match the image by setting the room size and fridge position.
>
> You can use the following properties. Each must be preceded with a '`--`' as the system uses CSS Custom Properties.
>
> Press 'Test' to try out your CSS in the environment to the left, and 'Submit Final Version' when you are happy with the result and want to move to the next step.

This task contained very straightforward closed commands to complete a simple task, while introducing the participant to the interface and concepts involved. It also introduced the problem as a scenario, which is important for later tasks.

5.4.3.2 Task 2

The second task presented the user with a more complex environment, a more open task, and a scenario without explicit instructions. The prompt for this task, again followed by a list of available properties, was as follows:

> When you get home from the shops you realise you accidentally left the hob on and the fridge door open. You also forgot to turn the oven on to heat up to 180°C ready for dinner.

> Luckily, your house is CSS compliant. This means you can make a style sheet that you can apply before you next go shopping which will both make you house safe, and turn your oven on to heat up while you're away.
>
> The CSS below shows the current state of the room. Edit it to make a new style sheet you could apply.
>
> This style sheet describes the current state of the kitchen. Edit it so that the room matches the image above, and sets the oven to 180°C.
>
> The end result will be a style sheet you can instantly apply to the kitchen before you leave, or remotely via your house's app.
>
> Tip: you may want to use 'iot-oven' as a selector

The description is deliberately open. However, there are still defined success criteria, namely setting the oven temperature, turning off the hob, and closing the fridge door. The user was still given an initial scene to work from.

5.4.3.3 Task 3

The third task was more open and did not present the user with an initial style sheet. The participants were still given a list of available properties, but the prompt had much less well-defined success criteria:

> Your company has just invested in IoT furniture which can automatically reconfigure a space depending on what it's being used for. It uses magnets or something.
>
> You've been asked to create a style sheet to set up the office for a normal morning. Jesse (desk 2) and Alyssa (desk 6) asked you to make sure their computers and lamps are turned on at the start of the day as they both like to get to work early and get working immediately.

This task was designed to discover whether the participants would be able to navigate the DOM tree for the IoT environment and understand the hierarchical nature of the environment.

5.4.3.4 Task 4

The final task was intended as a blank canvas for the participant to experiment with what they had learned. The prompt was:

There is no reference image or correct answer for this task. Be creative!

You've been employed by a venue to implement a style sheet for a concert they are planning. The organiser has sent over a list of requests, and the venue manager has sent you a list of controllable devices in the venue. Don't worry about fire exits, and assume the entire architecture is configurable around your design.

Below is an equipment and CSS property list from the venue:

... a list of furniture and equipment, with the properties that apply to each ...

Note: These have already been added to the DOM for you by a junior dev but they may have added some items you don't need.

Requirements, from the event organiser:

- The company brand colour is `#ff4422`, so we'd like to emphasise that on the stage.
- We'll have music playing initially, but we don't want it to overwhelm the guests, so please keep the volume fairly low.
- The stage will have a lot of equipment on it, so we'll need to keep the artists cool. However, if we cool the whole auditorium to the same level it will cost too much and the guests will be too cold.
- While we don't want the ceiling lights to overpower the stage lights, we have fire safety requirements that mean we need to keep the lights at the back of the venue at 80% or more.

Other than this, we will leave the layout and design in your capable hands.

The task was designed to be as close to a realistic scenario as possible. Although it is deliberately demanding, the expectation was that not every participant who started the task would produce a complete solution. It was hoped that their thinking and use of the system would be revealed to some degree.

5.4.3.5 Results

Results from tasks:

These results are based on the participants who completed each stage of the study.

The data on the number of attempts shows significant investment in the task by the participants who completed them. Tasks 3 and 4, the more complex ones, had a median number of attempts of 64.5 and 76, respectively. Each attempt was an instance of the participant entering a change to the style sheet then clicking the 'Test' button to see how it

Task 1	
Total participants/final attempts	46
Total non-final attempts	242
Total attempts by task finishers	286
Min attempts by task finishers	2
Max attempts by task finishers	18
Mean attempts by task finishers	6.22
Median attempts by task finishers	5
Task 2	
Total participants/final attempts	45
Total non-final attempts	346
Total attempts by task finishers	390
Min attempts by task finishers	2
Max attempts by task finishers	25
Mean attempts by task finishers	8.67
Median attempts by task finishers	7
Task 3	
Total participants/final attempts	35
Total non-final attempts	2661
Total attempts by task finishers	2611
Min attempts by task finishers	6
Max attempts by task finishers	215
Mean attempts by task finishers	76.76
Median attempts by task finishers	64.5
Task 4	
Total participants/final attempts	25
Total non-final attempts	1568
Total attempts by task finishers	1534
Min attempts by task finishers	5
Max attempts by task finishers	139
Mean attempts by task finishers	72.86
Median attempts by task finishers	76

TABLE 5.3: Results from all participants who completed at least one task.

Task 1	
Total attempts	156
Min	2
Max	18
Mean	6.24
Median	4
Task 2	
Total attempts	204
Min	2
Max	17
Mean	8.16
Median	8
Task 3	
Total attempts	1917
Min	41
Max	215
Mean	79.83
Median	65.5
Task 4	
Total attempts	1534
Min	5
Max	139
Mean	72.86
Median	76

TABLE 5.4: Results from participants who completed all four tasks.

affected the representation of the environment. All but the final attempt from each participant represents the participant completing a feedback loop from input to output and back again.

The mean is higher than the median on all but Task 4, meaning generally the data are positively skewed. The openness of Task 4 likely explains this difference, as there were no concrete success criteria given that the participant was free to stop when they felt they had done enough, and the task was more creative than the others.

As can be seen in Table 5.3 and Table 5.4, the mean and median attempts by those participants who finished all tasks were not significantly different from the entire population of the study who completed any number of tasks (i.e., a participant who completed three tasks behaved broadly in the same manner as one who completed four, irrespective of how far they progressed in the study). This adds some weight to the results from those that finished all the tasks and, while they cannot be extrapolated to larger populations without more data, they suggest that generalising to the larger population of study participants is not unfounded.

The relatively low numbers of attempts needed to complete the first and second tasks in the experiment, as well as the number of successes the participants had across all tasks shows that the participants were capable of completing them. The learning curve was overcome remarkably quickly and, hopefully, it was quicker than an unfamiliar system would have been, although this is largely conjecture without supporting data.

The recreations of the participants' scenes were made by replaying each attempt so that any bugs or inconsistencies were also replicated in the final version. The full versions can be seen in Appendix B.

Note: There was a bug found by several participants which meant that the lamps in Task 3 sometimes failed to turn on despite the CSS being correct. As a result, this has been ignored in the final results. The bug was caused by the `window.getComputedStyle()` not being called on the children of a changed element.

5.4.4 Part 2: Understanding Participants' Perception

The developers' initial view of an environment was tested by asking them to describe a photograph of a simple scene at the start of the experiment. This occurred before exposure to any syntax used in the experiment. They were given the following prompt, with a focus on the audience being similar to themselves:

> *The image below contains a scene. Please describe it as best you can in the text box, then press 'Submit' when you are happy.*
>
> *You should describe it in a way that another programmer like you would understand it. They should be able to replicate it from your description alone. Assume they will think similarly to you, and please frame your response accordingly.*
>
> *Please spend around 5 minutes on this task.*
>
> *This description can be in plain English, pseudocode, or using any other textual representation (except ASCII art or equivalent).*
>
> *Any code used does not have to be syntactically correct.*

After completing the four tasks above, they were again told to describe the same scene with a similar prompt to that before:

> *In the section below is the same scene you saw earlier, but this time please describe it in a way that another programmer who has completed this study*

> *would understand. They should be able to use your description to recreate the scene. Assume they will think similarly to you, and please frame your response accordingly.*
>
> *There is no obligation to use the syntax from this experiment. Please only use it if you believe it to be a good option. However, if you do, please feel free to extend it or alter it in any way you see fit.*
>
> *Please spend around 5 minutes on this task.*
>
> *This description can be in plain English, pseudocode, or using any other textual representation (except ASCII art or equivalent).*
>
> *Any code used does not have to be syntactically correct.*

This prompt emphasises that their target audience was the same, but that the people they are writing for would have also completed this experiment. The aim was to investigate whether their experience during the experiment changed the way they answer the prompt. It was expected that if the CSS-based approach was very acceptable, then the second time they described the scene they would have used a more structured approach, taking inspiration from the tasks. However, even if this were not the case, differences in their initial and final descriptions could reveal more about their experience of the experiment.

5.4.4.1 Results

Some 22 participants described both scenes and completed the tasks in between. The perceptual change observed between before and after the task is stark and can be seen in Appendix C. The initial descriptions varied wildly and are mostly written as either descriptive paragraphs or lists, with only a few written in some kind of pseudocode, despite the prompt explicitly allowing it. Of the second set of descriptions, however, many use the syntax in the study, or some variation of CSS syntax, and most show a heightened use of hierarchy to describe a scene.

Three participants showed clear engagement with the idea beyond the others. Two produced an entire DOM hierarchy for the room in the image, while the other extended the syntax to use a nested SCSS-style approach, including the use of SCSS variables.

Interestingly, several participants displayed a pre-existing preference for a hierarchy to describe the initial scene. While this is not unexpected, as a lot of code is written based on hierarchy and inheritance, it does help to verify an assumption of this thesis that developers will try to apply a hierarchy to a spatial problem.

Other interesting points included:

- The amount written in the first and second descriptions were often fairly similar for the same participant, yet varied greatly between participants. This is to be expected, as both tasks were limited to five minutes. Some participants clearly gave up and did not fully complete the second description.

- The details focused on in the first and second sections were very different for most participants. In the initial description many people mentioned the appearance of various parts of the room, including the colours of walls, the floor, the number of blades on the fan, and the contents of pictures on the wall. In the second description, they became a lot more intent on describing the position and layout with less focus on the details. This could be a result of the experiment asking them for simpler characteristics, often pertaining to position and layout, or an implicit assumption that the items in the room are already defined and that they can take a higher level approach. Fatigue could also have been an issue here.
- The CSS approach offered a lot less descriptive power than the free text descriptions, which were much richer in both detail and context. Achieving the same level of detail using CSS and the DOM would likely require linking to external resources such as images, and linking to semantic data about each object within the room.
- Even some of those who did not choose to use the CSS approach did move from an abstract description to using coordinates to describe positions.
- Some described the same information in both descriptions, despite using elements of the CSS approach in the second, although others completely changed the way in which they described the environment over the course of the study.
- One participant used an approach in the first description which approximated the way CSS works, including using 'position', 'width' and 'height' as they would be used in CSS rules. Another referred to 'padding' in the same context to how it would be used in CSS.
- Many described the room first before describing the contents, and some of those also described a piece of furniture first before describing what it contained. This is evidence of at least a basic hierarchical view.
- Some had a very disordered approach and described objects in the room seemingly at random, while others chose a sequential or hierarchical approach.
- Some people described the social context of the room (e.g. "a bedroom of a possible streamer") while most only considered the physical characteristics of the room.
- Many initially chose an arbitrary viewpoint in the first image, for example "from the perspective of the door", before going on to describe the room from that viewpoint. In the second image, many more people used the Cartesian coordinate approach. Those that used coordinates in the first image felt the need to describe the coordinate system before using it, but those that used it in the second assumed a shared understanding.

Many of the observations exhibit a lack of a common language for describing a scene in the first part of the experiment but, after completing the tasks, the participants made many more assumptions based on shared experiences. Many assumed a coordinate system and most assumed that the objects in the room needed less description to be able to reproduce the scene. It may be that the participants felt they had already described the objects in the scene so did not need to describe them again. However, it is still promising that they began to adopt the shared language of the system.

There is undoubtedly a lot more that could be read into the participants' responses. However, it is clear that a degree of learning and acceptance of the system occurred over the course of the study which gives weight to the use of this approach to satisfy the third aim of this thesis. It does however indicate that this approach could be acceptable to Web developers.

5.4.5 Part 3: Assessing the Acceptability

The primary measure of the acceptability of this approach was a simple questionnaire with Likert scales and free text questions. The questions, presented below, were designed to assess whether the participants subjectively liked or disliked the approach; whether they would actively engage with it; and whether they would introduce others to it given the chance. It was assumed that engagement and choosing to advocate the technology would be reliable proxies for acceptance. Participants were asked whether they would consent to follow-up research, in which case they could be contacted for clarification or further questioning.

Additionally, acceptability was assessed using the second scene description, as a tendency to use the approach, despite being told they do not have to, could indicate that it has become a preferred method for tackling this problem.

Due to the interactive nature of the Reddit website, some unprompted feedback was also received as responses to the recruitment posts, which added to the understanding of the user feedback from this study.

The questions asked were:

1. **How intuitive did you find the CSS-based approach to controlling an environment?** A 5-point Likert scale from very unintuitive (1) to very intuitive (5).
2. **Would you consider taking a job which used this technology (or a refined derivative of it) to program environments?** A choice was offered from no, maybe or yes.
3. **Would you contribute to an open source project which used this technology, or would you integrate it into an open source project you contribute to or maintain?** A choice from no, maybe or yes was provided.

4. **How likely would you be to use this approach in a commercial setting, were it to be specified and supported to the same level as other Web APIs?** A 5-point Likert scale from very unlikely (1) to very likely (5) was used.
5. **How would you improve this approach?** A free text field followed.
6. **Do you have any other thoughts about this project?** A free text field followed.

Note that question three focuses on an open source project, while question four specifies a commercial project as the two have very different processes and criteria for the selection of technologies.

5.4.5.1 Results

It is important to note that the participants who had access to this questionnaire would have been those who had managed to complete the tasks before this juncture. Moreover, they were also those sufficiently motivated to complete the entire study. This may cause the findings to be more positive than were all participants questioned in the same manner.

In an attempt to combat this, all the participants who had not completed the initial study and had indicated that they would be willing to be contacted to participate in future research were emailed and asked to complete the feedback section. Of these 18 individuals, two responded and completed the final questionnaire.

This questionnaire resulted in some very useful insights into the participants' opinions of the system, including many well thought out criticisms and compliments.

Much of the negative feedback pertained to the programming interface itself, including features such as autocomplete, or the UI in general. This is understandable, as the interface was very basic and those features which developers take for granted in IDEs take a lot of work to implement, and as such was not possible for this study. It did however reveal that there was at least a minimal interest in integrating the system with other technologies. There were other useful ideas and criticisms too, which will be addressed in detail in the discussion section as they apply to the thesis as a whole.

One element of the feedback, itself a response to the recruitment post in the css_irl subreddit [225], was the best example of engagement with the contents of this thesis. A person had both taken part in the study and was sufficiently interested to take the time to suggest an improvement, namely to use media queries to select when a rule would be active. Specifically, the individual said:

> *I think this is a phenomenal idea, and would love to be able to script my scenes with something like this. Added bonus, would love to see media queries for time of day and similar events (like sunset and sunrise).*

This idea echoed the idea presented in Mate Marschalko's blog post [190], which had been determined to be too complex in the implementation of this system. However, the concept opens up the potential for many more extensions, as explored in Chapter 7.

On a more quantitative note, those that responded to the questionnaire gave the following average responses:

How intuitive did you find the CSS-based approach to controlling an environment?

- mean: 3.58
- median: 4 (intuitive)

Would you consider taking a job which used this technology (or a refined derivative of it) to program environments?

- no: 4, maybe: 14, yes: 6

Would you contribute to an open source project which used this technology, or would you integrate it into an open source project you contribute to or maintain?

- no: 3, maybe: 11, yes: 10

How likely would you be to use this approach in a commercial setting, were it to be specified and supported to the same level as other Web APIs?

- mean: 3.42
- median: 4 (likely)

These results are promising as they all show at least a slight tendency towards the positive end of each scale. As mentioned earlier, this could be biased by the nature of the participants who reached it, particularly those who answered question one. However, the later questions serve as measures of confidence of the quality of the solution rather than mere utility, and those too have a strong bias toward the positive. This indicates that the system, with some obvious user experience enhancements, would be a viable candidate for adoption by the wider development community and, therefore, provide strong evidence that the third aim of this thesis has been satisfied.

5.4.6 Bias

There were at least two sources of bias in this study, aside from the author's own unconscious biases. These were selection and survivorship bias. Selection bias was both deliberate and a by-product of the method used. Requests for participants were placed only where they would be seen by those people with an interest in development and, specifically, in Web development. This was because it was not believed that a normal user would have been able to complete the task within a reasonable time, if at all, and because developers are the target demographic. The recruitment posts and study were both in English, thereby excluding any non-English speakers, as the author does not speak any other language. The task itself required a fairly modern browser, therefore excluding those without access to a modern computer. The study was also not accessible to non-visual browsers such as screen readers. Addressing these issues was beyond the scope of the study, but should be assessed were wide-scale adoption to be considered.

The tasks within the study were a major cause of survivorship bias, as they were both long and complex. This is shown by the higher drop-off rate as the tasks increased in complexity. Emailing those who abandoned the study was an attempt to negate this bias, but was not very successful as many participants had not consented to further communication and only two of the other 18 responded. Comparisons of the population of those who began the study with those who finished it did not show any significant differences in the parameters measured, and so there is no way to know for sure the cause behind the drop in participation over the course of the study. Two participants wrote notes to apologise for not having sufficient time, and it is a reasonable assumption that many others did not complete it because they also ran out of time. Although, alternatively, participants may have reached the limit of their attention or skill.

Chapter 6

Discussion

This research has investigated several approaches which increasingly take advantage of Web browser technologies for describing, monitoring and controlling IoT environments. Each approach has compromises, yet each subsequent iteration has become less proprietary. The reason for this is two-fold; firstly, exploring the problem afforded greater knowledge of the problem space and what was required; and secondly, over the period the project took place, browser technology and Web standards advanced significantly. Specifications such as Web Components and Custom Properties have made the browser far more adept at representing non-document elements. While the solutions became increasingly aligned with how browsers are implemented and designed to work, the browsers themselves became a more hospitable environment.

This project intended to cater to the IoT without merely producing a Web service that is accessible to the user, but rather to envision a way that the IoT and, by extension, the physical world becomes a part of the Web and meshes with the documents the user sees every day. In doing so it goes a step beyond previous visions of an integrated WoT, including that of Guinard, Trifa and Wilde [192], which were content that connecting to devices with Web protocols such as REST was sufficient. Concerning this, the project has been more successful than originally expected.

6.1 Have the Aims Been Met?

6.1.1 Aim 1: To treat IoT devices as we do the elements of a Web document by representing them within a DOM

The first aim of this thesis was to find a way of representing IoT devices using the DOM. While this could have simply been a list of elements in an XML document imported into an iframe, it was not felt that this would be sufficient to make the IoT truly integral to the Web.

Instead, a hierarchical approach was explored, one which progressed from XML to HTML and, ultimately, allowed for the inclusion of digital twins of the devices directly within the DOM of an HTML document. In this respect, both the letter and spirit of this aim have been met.

Experiment 1 laid the foundations for a DOM-based WoT system, and this was expanded on in Experiments 2 and 3, each of which added an extra dimension to the solution. Experiment 2 demonstrated that it was not only theoretically possible, but also feasible to build and execute a WoT system using the DOM as a core component. Experiment 3 took this a step further by using Web Components to begin to add semantic meaning to the DOM nodes which were the digital twins.

In every experiment, a DOM node represented a device, although the positioning of the node within the hierarchy added meaning beyond the device itself. Most widely employed was the idea of a hierarchy to represent a physical environment, wherein an item on a table becomes a child of the table node. However, the potential for expanding this beyond purely physical representations was also touched upon.

6.1.2 Aim 2: To build a system for controlling and monitoring IoT environments using only browser technology, ideally with CSS and JavaScript

Similar to the first aim, there was both a shortcut solution that could have been taken and a more honest approach that was taken. The shortcut to building an IoT system using browser technology would have been to heavily rely on JavaScript to build new functions and structures that would run within a browser, but not follow the best practices of the Web. To a certain extent, the system in Experiment 2 took this path. However, where possible the built-in features of the browser were taken advantage of and every effort was taken to not misuse them, but rather to implement them in a way that would be expected to a Web developer. For example, in Experiment 3, the Web Components specification was used to represent devices and their components rather than taking a simple approach of representing the environment in a JavaScript object with links to HTML elements on the page. A comparison of these can be seen in Figure 6.1. Both meet the requirement of using only browser technology, yet Web Components work with the browser while the other approach works within the confines of it, although ultimately overlooks integration with it.

Despite every effort, there were some aspects of some browser specifications which had to be worked around in each experiment. In the first experiment, many opportunities for integration were missed by seeking an XML approach rather than an HTML one. While the XML was valid and would likely have succeeded from a technical standpoint, it nonetheless over-used attributes and would not have worked well with other technologies such as CSS. This was taken as a learning point early on and informed the design of Experiment 2.

```
const environment = {
    devices: [
        {
            name: 'cooker',
            element: document.querySelector('#cooker'),
        }
    ],
};
```

```
<div id="cooker"></div>

<iot-cooker />
```

FIGURE 6.1: A JavaScript object linked to a `<div>` element (top), vs. a Web Component (bottom).

The second experiment suffered from having to work around the CSS property allow list. CSS only permits certain approved properties such as ‘color’, ‘animation’ or ‘height’, while any WoT system would require far more freedom to represent the physical states of devices. A separate parser had to be written to handle the new properties, thereby bypassing many built-in features and optimisations. This also meant that the approach was abusing the flexibility of JavaScript to fill a gap that should have had a native solution. This was solved in Experiment 3, as browsers gained widespread support for Custom Properties, which allowed for using properties such as `--position-z` and `--power` which were not possible in Experiment 2.

The third experiment had the fewest compromises, yet was still very slightly imperfect. It relied on an event being triggered on a node when its CSS properties changed, which is not a current feature of the browser. An example of this is when the power property of a lamp element was set to `on` and this change to the CSS would then trigger an event to change the appearance of the digital twin. The same event would have been used to trigger a message to the device itself in a real system. This change to the property could have arisen directly through the `style` attribute or else via a class applied anywhere above the parent. This is the equivalent of a repaint triggered by a class change that affects the element in question. However, the browser does not expose this event to the JavaScript engine. Were the event to be exposed to the JavaScript engine, then the solution would have been almost entirely native and inline with the spirit of the browser’s specifications. However, as it stood, an inefficient workaround was necessary to poll for changes to the CSS using `getComputedStyle`.

While this aim was not entirely met in practice, it was demonstrated that it would be possible to meet it with some very minor adjustments to the browsers’ implementations. That said, even minor changes would require significant justification and work to implement due to the scale of the social impact of an API or change in functionality to mainstream browsers. Overall, it was considered that despite this, the aim was met as two functional systems and one theoretical approach were produced which would satisfy it.

6.1.3 Aim 3: To produce an approach which is acceptable to existing Web developers that could allow them to easily transition into WoT development, thereby following existing best practice for Web development

Each experiment brought the solution closer to a standards-based approach and, with this, it also came closer to best practice for Web developers. While not all best practice follows standards (e.g. frameworks such as React often work around the standards or overlay them with a completely new paradigm), all standards form best practice. Once a standard is implemented in the browser's native code, more often than not it becomes the most efficient way to complete a task and, by default, becomes best practice.

From a more human perspective, Experiment 4 demonstrated that there is community support for this type of approach, with several developers taking the time to explain what they liked about the approach, while others were sufficiently engaged to share feedback about both the approach and the UI. Most developers who attempted a task were successful in completing it and all developers produced an answer for each task they attempted which used the approach correctly, whether or not the end result was correct. This is strong evidence that the approach is relatively straightforward to learn and adhere to, which suggests that the developers who took part could relatively easily transition into using this process for WoT development.

The feedback at the end of the study revealed a slight leaning towards positive sentiment when the participants were asked about the approach used, as many would recommend it to others or use it themselves in future projects. While this is by no means conclusive, it suggests that developers may willingly accept the system were it to be developed further.

There was also some evidence that the developers quickly learned to think about environments hierarchically, even if they did not appear to before the study commenced. The change from an unstructured to a structured, hierarchical description of the scene at the start and end of the tasks in Experiment 4 reveals a change in their perception brought about by the study. Again, this is not a conclusive result, but it is yet another indication that the participants could quickly adjust to the way of thinking necessary to use the system.

It is worth noting that the developers who took part were those that responded to the initial invitation and were sufficiently interested to complete the study, while those participants who showed support were those who successfully made it to the end of the experiment. This means that they had passed the initial explicit and implicit filtering processes of the study. This is not necessarily a problem, as every development community has filters created according to interest, knowledge and experience, and a potential community around integrating the IoT with the Web would also evolve equivalent selective biases. What Experiment 4 did successfully demonstrate was that at least some existing Web developers would be able to transition from Web development to WoT development and, moreover, that they would do so willingly, potentially advocating this transition to others. There is more work to be done in exploring the level of acceptability and what can be done to facilitate it, although it can tentatively be said that this approach is acceptable to some Web developers in the form presented in Experiments 3 and 4.

6.2 Benefits of This Approach

Many benefits derive from this approach, and these overcome some of the limitations of existing systems. The approach, as a whole, is far closer to Web standards than the IoT ecosystems from blue-chip companies such as Google [143], Apple [127] and Amazon [145]. Matter [156] is forging ahead with proprietary integrations between manufacturers, and even the W3C's WoT implementation requires a great deal of new technology to function. The experiments in this thesis demonstrate that there is a way to implement a WoT system without creating too much new technology.

In addition to the benefits gained from completing the aims of this thesis, there have also been some additional benefits that surpass the capabilities of other IoT systems, including zero-knowledge environments, integration with Web documents, and style sheets for environments.

6.2.1 Zero-knowledge Environments

Experiment 2 was the first system to realise the idea that no single part of the system needs to know about the rest of the environment, and this was enabled by the use of CSS classes. Even the hub, which contains a DOM of all the devices does not necessarily need to know about all the components of the devices, as messages were routed to the devices; yet acting on them was delegated to the devices themselves, and the components that were acted upon may or may not have been visible to the hub. Indeed such a device could have had its own equivalent of a Shadow DOM.

This type of system comes with some useful advantages, notably, that any class-based action applied to the system is agnostic of the environment it is applied to and can operate without any implicit knowledge of the contents of its environment, i.e. it is weakly coupled to the environment. For example, a style sheet can be written generically and applied to any environment. In this case, only those objects which meet the criteria of the selectors will be acted upon, and those rules which are not used are ignored. The same cannot be said for many other IoT systems, including IFTTT [133], Amazon Alexa [145] or Node-RED [134], which require functions to be written to specifically link actions to devices. In these tightly coupled systems a function in one environment cannot easily be shared with another, unless they are almost identical.

A loosely coupled IoT system and environment yield a far more resilient solution. Considering a hypothetical environment: if a device lasts for an average of three years, then in an environment with 1,000 devices, one will break critically almost every day, assuming a uniform distribution of failure. This, of course, does not include routine failures such as battery failure or loss of connectivity. In an environment with significant complexity, flexibility is therefore imperative. A loosely coupled system allows for failures and replacements and, provided the replacements have the same classes applied to them, it will continue to work seamlessly, regardless of any differences in the underlying hardware. It also allows for expansion (e.g. replacing a single radiator with two smaller ones), without any change to the style sheet.

There are of course downsides to a loosely coupled system, and the most glaring is that two devices may share the same classes, yet have very different functionalities, especially in the case of homonyms. A device with a class of `light` may be a light or it may not weigh much. Turning on this second device may present issues if it is potentially dangerous, such as a light camping stove.

6.2.2 Integration with Web Documents

Experiment 3 showed the power of fully integrating digital twins into a document wherein the representation of the environment was generated from the twins themselves. A benefit of this approach is that the representation of the environment is always up to date as, if a device is removed, then its twin is also removed and the representation is updated by default.

The twin becomes woven into the fabric of the document and so the data it contains becomes a part of the document rather than using scripts to display data about the device. This is another example of how the WoT can mirror the Web, as the concept of data fading into the background is a key feature of Marc Weiser's notion of calm technology [116]. A simpler, yet more deeply embedded example of this, is a weather station in which the digital twin is embedded in a user's browser homepage so that they can see the current external temperature, humidity and wind speed. The outcome is arguably similar to that of using a script to call an API, but with a tighter bond between the device and the document. Using a similar approach, it would be possible to construct an interface identical to the Web UI used in Experiment 2 but built from digital twins.

This tight integration is something that lies beyond existing IoT and WoT systems, and only has brief parallels to the Amalgam project [2].

6.2.3 Style Sheets for Environments

The use of style sheets is very effective at reducing repetition, but also brings to the IoT a feature that is analogous to visual themes. This was briefly explored in Experiment 2, but only with small style sheets to swap between scenes. In this case, the scenes had a purpose such as ‘Reading’ or ‘TV’, and controlled multiple devices using the zero-knowledge approach mentioned previously. This is a parallel to the themes offered by some Web sites and applications, in which users can switch to a different theme. This can be done for the purposes of self expression; to cope with a change of environment (e.g. a dark theme at night); or to help them to overcome a disability, for instance a high contrast theme. WinAmp [174] was famous for allowing its users to download themes and flip between them, and Apple MacOS and iOS have lead the way recently in encouraging developers to include dark mode themes [226].

With style sheets for the IoT, users can replicate this functionality in physical environments. Rather than separately specifying each device on every occasion, they can use a pre-defined style sheet and switch to it with a single button press rather than via a complex manual process. This is similar to what is offered by Philips Hue and some high-end smart home systems, including Savant [160] and Control4 [158], but these are neither generally applicable nor produced in a standards-compliant way.

6.3 Issues With This Approach

While the approach has many benefits, the experiments also raised several issues which were not resolved in this thesis. Three of the most important were the CSS property taxonomy, functionality and permissions, and industry support.

6.3.1 Taxonomies

Experiment 2 highlighted the issue of a CSS taxonomy as it required significant work to overcome the CSS allow list for property names. New properties had to be chosen and added to a new allow list within the hub. The decision about which words to choose was executed as they were needed, rather than in any structured way. Experiment 3 overcame this by allowing any property name to be used, but this opened the system to the possibility of spelling mistakes, which were often made, and a sprawling folksonomy of property names. Neither situation is ideal.

The current list of CSS properties was curated by the W3C and browser vendors to match a relatively small number of possible states. The same approach would be challenging in the IoT due to the heterogeneity of the devices such an environment could contain. New properties had to be added frequently during Experiment 2 to allow for new situations, which could have been hard to predict before they arose. A closed taxonomy was shown to be insufficient, but opening it was not much better.

Experiment 3 had an open taxonomy, one which did not allow for error checking for the names of properties. As a result, misspelled properties frequently appeared and took a while to find. As these were not errors of the system itself, they could never be flagged by it. Again in this experiment, properties were added as they were needed. This approach required less overhead for development, but led to a larger taxonomy than ideally required. Both spelling mistakes and synonyms enlarged the taxonomy unexpectedly and had to be manually corrected.

6.3.2 Functionality and Permissions

CSS, as it is today, assumes that all fields are both readable and writable. However, this is not always true for an IoT deployment. It may be that a position of a device can be read, but the device lacks any actuators which it could use to move and change its position, meaning that it cannot be written to. This is an issue of functionality. However, permissions may also incur the same consequence. A user may want to apply a style sheet, yet not have adequate clearance to do so. For example, they may want to open all the doors of a building, but lack the permission to open a fire exit or a locked room.

CSS does not have any concept of this type of property, and this is a potentially a serious, though subcritical issue with the approach used in this thesis. Any style sheet designed for re-use or use by different users would have to make certain assumptions about the interactivity of an environment. If a style sheet were a safety-critical one, such as unlocking all doors and facilitating an evacuation before using a CO_2 fire extinguishing system, it would be catastrophic if it assumed the door locks were automated when they were not, or that the user who initiated it had permissions which they did not.

6.3.3 Industry Support

A large and socially complex issue is that of support in the development industry. Experiment 4 touches upon this and found that developers may be willing to advocate the approach in future. However, this is a small part of a much larger process. As AXR [68] found, support does not come because of a unique or even a good idea, but rather derives from this in combination with hard work. Matter is seeing success built from overcoming the competing agendas of large corporations, meanwhile the W3C's achievements come from verbose collaboration between individuals and their large body of extant work.

The approach here stands a chance because it is so closely linked to how Web development is conducted today and, therefore, requires little change to workflows and does not need much learning to use it. As Experiment 4 found, Web developers were able to use it almost instantly and, within an hour, most had adapted their thinking to match the hierarchical modelling required to use it.

However, as with all projects mentioned throughout this thesis, reaching a consensus within a group of IoT experts takes many years, while implementation takes even longer. Agreement between groups, as shown by the lack of cooperation between Matter and the W3C, is even harder to achieve and most likely falls beyond the remit of the work carried out in this thesis.

6.3.4 Why Not Just Use JSON Instead of CSS?

An additional issue and one which was raised more than once during the project, particularly by one study participant, was the suggestion that JSON might be better for representing devices than DOM objects. Hopefully, the content of this thesis implicitly answers this question. However, it has been so prominent that it is worth addressing specifically. It would be possible to represent a device in JSON, JavaScript, or several other object-based data structures. The CSS properties could ultimately become JSON properties, while the selector could be stored as another JSON property, and it may even resemble Figure 6.2. However, that is where the utility ends. Beyond that, it would not be able to take advantage of many of the browser features which CSS can. For example, the browser has a very advanced engine for calculating cascades and applying CSS rules to DOM elements. Using JSON would mean that this has to be remade or else the existing engine would need to be significantly altered. This is possible, yet unfeasible. It also goes directly against the core purpose of this thesis, namely to explore re-use over new solutions and workarounds.

```json
{
    "selector": ".kitchen .ceiling.fire-alarm",
    "power": "on",
    "volume": "85dB"
}
```

FIGURE 6.2: A JSON representation of a fire alarm.

The use of JSON would bring with it certain benefits, such as the ability to quickly modify the rules using JavaScript and being easily serialisable for transmission, although these benefits are not significant to the aims of this thesis and come at too high a cost to be considered worthwhile. As mentioned previously, this would again be an approach working within the confines of the browser, rather than working with the browser.

6.4 Issues That Have Not Been Directly Addressed

Security and scalability are two issues which are integral to the IoT, yet have been largely disregarded in this thesis. They were deemed to be beyond the scope of this project, but it would be remiss to entirely ignore their potential impact. While they are both important, their impact on the contributions in this project were minimal. Security, at least in a technical sense, was handled by the underlying Web and Internet technologies used. Meanwhile, both scalability of networks and of the DOM are subjects which are orthogonal to the concepts put forward here. That is to say, the approaches used here rely on solutions to security and scalability, but do not seek to solve those problems within this thesis.

6.4.1 Security

IoT security is a field in of itself and, in general, it was assumed that any system based on Web technologies would both be able to take advantage of existing and future Web security and that it would be relatively simple to implement security later on. This is a broad assumption, but one which is necessary to allow the freedom to produce the experimental systems used. The only concession here is that of the discussion of permissions presented earlier in this chapter. However, that discussion was deliberately superficial as it falls beyond the bounds of this project.

6.4.2 Scalability

Scalability is a concern of any IoT system, although much less relevant in the experiments in this thesis. Large scale IoT deployments are very dependent on their underlying infrastructure and, as such, latency, bandwidth, addressing, and a large number of other limitations must be carefully planned for and overcome. In this project, the deployment of Experiment 2 was small enough for network limitations not to be a concern, and Experiments 3 and 4 abstracted away from a physical network entirely.

This particular WoT implementation was bounded by the capacity of the DOM to represent individual nodes. While there was no indication that this limit would be practically reached in most deployments, it would be a factor for consideration in a real-world environment. In the cases of the experiments here, none came close to the limitations of the DOM, even on fairly restricted computers.

While Experiment 2 did suffer from some minor latency issues, Experiment 3 showed that this was unlikely to be a result of the DOM or the Web technologies used with it. The Web, in general, and the DOM specifically have significant proven scalability. Therefore, any tests of the approach outlined here would be a test of the DOM implementation in the browser running the test. It is already known that modern browsers can cope with millions of elements in a single Web page, and so it is a reasonable assumption that they could also cope with millions of digital twins within a DOM representing an environment. The only aspect which was not scalable in Experiment 3 was the missing event when a component was redrawn, although it was assumed that this event would later be added to the browser, were this technology to be advanced to use outside of the laboratory.

The scalability of a hub or communication with devices also falls outside of the scope of this project. There continue to be many advances in this area which have nothing to do with the representation of an environment or resolution of raw commands from CSS rules.

6.5 Limitations of the Research

The limitations of this project fell into the categories of deliberate and/or unavoidable. The experiments were deliberately limited in scope, as mentioned previously, but were also products of their timing. Experiments 1 and 2 were completed before Web Components and Custom Properties were made widely available, and so could not exploit these very useful technologies. However, in some respects at least this is a blessing, as it forced creativity in the solutions and meant that more alternatives were explored and further problems highlighted. It also demonstrates the fast-paced nature of the development of Web technologies and how the growth of the Web can affect the growth of the WoT.

The fidelity of all of the experiments also limited their effectiveness at times. All were experimental and temperamental research grade systems, yet they still each provided valuable insights and sufficient information to both draw conclusions and inform the next experiment. The reliance on hardware limited the scale of Experiment 2, although enough different devices were produced, while Experiment 3 demonstrated that scale was possible.

However, the biggest restriction came in evaluating the solutions produced as there exists a lack of equivalent implementations. As seen in the literature review, those that exist are either research projects, built for evaluating specific features; large, well-funded commercial projects; or open source projects with many contributors.

None use a similar hierarchical approach for mapping environments and most send messages between individual nodes, which are people or devices, or from an individual to a group rather than employing the set-based approach taken here. This made direct comparisons of features ultimately futile, although there are some overlaps with other systems which have been considered in the next section.

Similarly, any comparisons of concrete metrics such as speed or availability are also meaningless, as large projects have had many thousands of hours spent on these aspects, and research systems do not need to demonstrate this unless it is a factor within the research itself.

The solution was to adopt a more subjective approach in evaluating the implementations, using both assessments and observations. Both the tasks and assessments were made by those with experience of Web development, and many of those were experts in the field.

6.6 Relationships to Other IoT Systems

The closest approach to those in this thesis is the one produced by the W3C's WoT Group [12], yet it is sufficiently divergent that they would never be in direct competition, meanwhile preventing the drawing of direct comparisons. Where the W3C seeks to describe devices and provide an API for discovering and interacting with them, this research provides structure and hierarchy to the discovered devices and provides a means to select those with which a user wishes to interact. The W3C's implementation stores the state of a device using a digital twin produced from a template TD [87], while this project enables set-based control of that state using classes and style sheets. While both produce digital twins, the W3C's digital twin could be represented as a DOM element using the work demonstrated here.

When this project started, it seemed very much at odds with the W3C's WoT implementation but, as both have progressed, they have aligned much better than expected. Initially, the W3C's WoT Group wanted to produce a way of describing devices, the 'Thing Description'; and a way to script devices known as the Scripting API. The Scripting API did not receive an outline draft until 2017 [227], but eventually came to include a runtime that implemented the WoT system, plus a WoT browser/discovery feature. This direction was at odds with Experiment 1, where devices were manually added to the DOM, and also with Experiment 2, where devices registered themselves with a server rather than being discovered. However, Experiment 3 is much more compatible with the W3C's specifications. As their specifications became clearer and this project's scope narrowed from practical issues such as communication to higher level issues such as orchestration, an alignment of the two became easier. The result was that Experiment 3 could contain devices linked to a WoT TD using an attribute (similar to a `src` or `href` attribute), and the discovery and messaging could be handled by the W3C runtime. Once discovered and registered, the devices could be added to the DOM manually or automatically, represented with the Web Component-based framework, and controlled via CSS. The TD could provide filtering for those messages which the device can and cannot understand and also offer hints in an IDE for the expert user.

Provided the Matter project progresses as expected and the W3C is correct in its assumption that this and the W3C WoT implementation will be compatible, then the work presented here should be able to act as a layer over both. However, the aims of Matter, the W3C, and this project are quite disparate.

Commercial systems are yet more distant, and most provide solutions with simple UIs which are ideal for the average consumer, like Amazon Alexa or IFTTT. The approach outlined in this thesis permits fine-grained control for an expert user or developer, which sits at the opposite end of the scale to the simplified UIs in many current IoT systems. While a simple UI can be created using the systems built in this project, the reverse is not possible: fine-grained control cannot be achieved with the blunt edge offered by many commercial systems. The notable exceptions are those more bespoke systems, including Savant and Control4, but these are closed source and not available to the average consumer.

Open source IoT systems such as Home Assistant [131] and openHAB [130] can provide an expert system through offering unlimited customisability, an orientation which is closer to this project. However, none so far offer a hierarchical approach to orchestration that is as complex and flexible as this one. Home Assistant allows users to group devices by physical location [228], while openHAB offers a more complex semantic structure [229]. Both of these could be used to represent devices in the same way as this project, although neither actively encourages it.

Academically speaking, much of the research surrounding the IoT and WoT has related to practical problems. Outside of a wider integration with the Semantic Web [96], very little has been done to structure the representation of environments in a Web-friendly way. This could be because existing solutions are sufficient for even the largest modern IoT deployments and so there is little drive to go beyond this. Publish/subscribe models which use a channel-based approach, such as MQTT [113], and even direct messaging have not been listed as a limiting factor within any modern IoT deployment in the literature. For the moment, network speeds and collision detection, processing power, and battery capacity and efficiency seem to be collectively outpacing the growing scale of IoT deployments. However, perhaps if this ceases to be the case in future, there will be a need for a tree-based approach such as this along with the efficiency gains that come from both the structure itself and the millions of hours of work that developers have put into optimising DOM traversal in the browser.

6.7 The Importance of This Research

This research is important because it diverges from extant research and technologies being developed elsewhere although it still follows Web approaches and standards. While it could prove to be a technological cul-de-sac, one that is never followed by mainstream technologies, it is a valid exploration nonetheless. Without experiments like these, alternate ideas would never be explored and we would forever be limited to the local maxima of the technological landscape.

Approaches such as those explored within this thesis are specifically important because they enable developers to apply the knowledge they already have to a new domain and this, in turn, could lead to short cuts in progress. In this case, a knowledge of how to design and create virtual documents can be applied to designing and creating physical environments, and Experiment 4 exhibited some success in this area. The same mental models, and even the same language and syntax, can be extended to allow for new goals to be achieved, without having to build new systems from scratch. The idea that we can re-use the discoveries and work of others is essential to development in all endeavours of human society. Transposing skills from one area to another could save time relative to nascent approaches that require new learning, and that saved time could be used to push the limits still further. Again, Experiment 4 showed that experienced developers could learn to use the syntax of the system quickly and could adapt to the process of thinking necessary to be successful in using it.

On a more tangible note, being able to adapt existing tools in the growing IoT space would benefit businesses both on the supply and consumption sides. Suppliers would be able to apply a few changes to their existing products to allow for IoT use at a fraction of the cost of building new products. The consistency of experience could allow designers and developers on the consumption side to quickly pivot to a new domain without the financial cost of retraining or the mental burden of learning a new perspective. While revolutionary ideas may be easier to sell, evolutionary approaches also have their place within the IoT landscape.

Chapter 7

Conclusions and Future Work

This chapter revisits those benefits realised and the issues raised throughout the course of this project, and goes on to theorise as to where future research may lead. The previous chapters have analysed the successes and failures of each experiment and the project as a whole, and concluded that it has achieved the majority of what it set out to do. Here, the common threads and unexplored opportunities are itemised and examined. To do this, each was placed into one of three categories; namely applications of the technology, extensions to it, or problems with it.

7.1 Applications

The experiments in previous chapters explored a select few applications of the technology and these highlighted some exciting opportunities, although they also revealed that the potential is far greater than the illustrations employed so far.

7.1.1 Hybrid Documents

Experiments 3 and 4 used a system that embedded the digital twins into the DOM to great effect. This 'hybrid document' concept of physical and virtual elements indicates the border between the Web and the real world breaking down slightly. The document sits on a computer within the environment, but the environment also sits in a representation within the computer, and thus each may act on the other with elegant symmetry.

The application of this principle in Experiments 3 and 4 was to use the digital twins to create the representation of the environment and, briefly in Experiment 3, to create an interface to the devices by making them clickable to control their state. Rather than having a separation between UI and the twin, the twins themselves became the interface.

In a hybrid document, the elements can contain presentational HTML within their shadow DOMs, which does not affect the overall functionality of the system, although it could conceivably contain control elements such as switches and sliders. In a fully implemented system, it could be that the UI templates and digital twins merge to form an interface that fluidly changes to represent not only the state, but also the composition of an environment without any requirement for developer interaction. Were a new device to enter the environment, it would register with the system, appear within the DOM, and automatically be rendered within the UI. This UI would be available on any client device that contains the DOM and a user could then view its state and control it, in the manner of a universal remote control that adapts to the devices present. In a busy environment, this could be somewhat chaotic, but the creation of a usable dynamic UI that is acceptable to users would be a worthy research goal.

The same hybrid device concept could be taken in another direction, as was briefly explored in Experiment 2, wherein CSS can be applied to both the document and the digital twins within it and, by proxy, the devices themselves. In Experiment 2, one of the pages in the Web UI was assigned the same class as an RGB lamp in the room and thus a single rule, one that targeted the appropriate class, was applied to both concurrently. This opens the door for creating immersive experiences between the Web and the environment the Web is experienced within. Beyond virtual reality, a user's actual reality could be affected as they explore the Web; a kind of augmented reality that does not require a screen and camera.

Besides the challenges of implementing such an approach, there are a number of human factors that would need careful consideration, least of all being the question of whether someone would actually want their house to respond to the documents they view. Additionally, the security and potential to exploit such a system would need significant attention. Given how commerce has affected the Web, there is a great deal of potential for even more invasive experiences with the WoT.

7.1.2 Themes

Linked to hybrid documents is the concept of themes. In this case, a theme is a curated configuration file that can be applied to a space which will bring in some manner of continuity, much like 'dark mode' for a UI makes it suitable for use in low ambient light environments. Philips Hue [141] has scenes which serve as limited examples of this, although the freedom and control of the DOM and CSS, coupled with the zero-knowledge environment, would bring significant improvements to this concept. These were demonstrated in Experiment 2, wherein style sheets were used to control multiple devices for different activities; an example can be seen in Figure 7.1. As mentioned in the previous chapter, style sheets for environments could be created by users for their own use, but beyond this, they could also be shared.

For a movie:

```
.ceiling_light {
    --power: off;
}
.television {
    --power: on;
    --brightness: 70%;
    --contrast: 85%;
}
.hifi {
    --power: on;
    --volume: 65%;
    --source: hdmi;
}
```

For a dinner party:

```
.ceiling_light {
    --power: on;
    --brightness: 60%;
    --color: sunnyday;
}
.television {
    --power: off;
}
.hifi {
    --power: on;
    --volume: 20%;
    --source: url('./dinner-party-mix.m3u');
}
```

FIGURE 7.1: Example style sheets for different scenarios.

In the early 2000s, MySpace [230] became an extremely popular social network. One of the key reasons for its success was that it not only allowed users to create a network of friends, but also they could have a personalised homepage which offered customisability, including applying themes to sections or the whole page. In what would today be a nightmare of cross-site scripting vulnerabilities, users could embed CSS and HTML directly into their page and it would be shown to all visitors. While some users were able to make their own, the vast majority used themes they found on various theme sharing Web sites. WordPress [231] has also gained a substantial market share because it allows relatively inexperienced users to apply expert-created themes to their website. It allows administrators to add CSS, HTML, JavaScript and PHP [232] to the server to customise the appearance but, as it is based on a standard API and page structure, it is relatively simple for creators to make themes and subsequently share them with less skilled users. Both MySpace and WordPress enabled large movements of both hobbyist and commercial users, creating themes for sharing within online marketplaces. The same approach is conceivable for the WoT and the IoT.

The technology in this thesis allows for a similar experience of providing themes. A style sheet could take advantage of the zero-knowledge environment of this WoT implementation and, therefore, would not have to reference specific devices or components by ID. It would be very easy for a developer to make a style sheet which targets very generic class names so that it could be used across a vast array environments. This style sheet could then be shared within groups, perhaps within an organisation to apply branding, or else between home users who want to have the feel of a curated space but lack the necessary skills to achieve this themselves. They could have a colour-based theme for their favourite sports team, or a sound and light-based theme to use when getting ready for a night out.

As the style sheet can be class-based, it would not necessarily need to be modified if devices were to be added to or removed from the environment. Provided that a new device was assigned class names that fitted within the existing schema, it would receive the correct commands when control messages were sent to the environment. This configuration was briefly experimented with in Experiment 2 and Figure 7.1 shows two examples of very generic style sheets.

The research potential extends into both technical and human domains. There is much that could be done to make environments more conducive to being styled, for instance through the use of shared taxonomies, although this presents its own problems as discussed later in this chapter. There is also plenty that could be researched into how to share style sheets both efficiently and securely.

On the social side, it would be interesting to learn whether people would be interested in such a concept, and also how frequently they would use them (perhaps daily, only on special occasions, or more unpredictably). Watching the propagation of themes through communities and recording how they might evolve, both as users become more accustomed to the idea and environments become more complex, could yield interesting insights as to how people actually employ IoT environments. Commercially, it would be useful to know whether people would pay for themes and treat them as consumable content, much as expansions for games are consumed now and mobile phone wallpapers were purchased in the early 2000s.

7.1.3 Composed Devices

‘Composed devices’ are those devices that do not physically exist as a single device, but rather are constructed from a combination of other devices or the components of multiple devices. Experiment 2 included a climate control system which exemplifies this; several devices, including a heater, fan, dehumidifier, and others, were combined and presented as an integrated ensemble. This is similar to how air conditioning systems are currently presented, as they too are a group of devices with a single interface which, to the user, appears to be a solitary device. This type of model has the benefit of abstracting the user and developer from the constituent devices within the system, allowing them to focus on the outcomes (i.e., a user can set the desired temperature rather than having to set fan and pump speeds individually).

The unique situation presented by the WoT implementation employed in this thesis is that it allows composed devices to be made from the components of other devices and not just the devices themselves. Every device exposes not only itself, but also its components to the DOM. This means that the value of every IoT device is multiplied as new devices are added to an environment because they not only have their own function, but can also be combined to form other devices which are more complex than themselves. The use of the DOM and in particular registering device components in the DOM allows for creating composed devices from both whole devices and individual components using a Web-native approach.

This approach is not restricted to just devices, as it can also include data sources or documents. Taking this approach, it would be trivial to create a composed device that integrates with physical devices and data to create a very pervasive experience. A key benefit is that the devices themselves do not have to be designed to be a part of it, or even consent to be used for a chosen purpose, and so a user may avoid the burden of extra cost, maintenance, and a separate deployment. For example, a user could link their phone to the power light on their television so that they could comfortably watch a movie while their phone remains silent, yet still be alerted to any urgent calls. This would give the user many options as to how they want to experience both their own data, as well as information of interest from the Web and other external data sources. They could choose to implement an approach conducive to calm technology or indeed any other approach from the pervasive and ubiquitous computing movements.

The situation presented by the WoT implementation employed in this thesis is one that allows composed devices to be made from the components of other devices and not just the devices themselves. It is unique in that the use of the DOM allows for complex devices to be built from simple queries using CSS selectors. It also allows composed devices to be made which contain both physical devices and elements from Web documents interchangeably.

Another Composed Device

Another example of a composed device would be an occupancy and activity measuring system integrating the power usage of various other devices within a home. Where devices in a room collectively measure an increase in power consumption it could be assumed that a user is either present or otherwise conducting an activity in that room. Combined with other sensors in cupboard doors, chairs, and other devices, it may be possible to get a very accurate image of occupancy without specifically purchasing an occupancy detection system.

How well this would work in practice would be a large research area in of itself, and there would likely be limitations in terms of interoperability, notably concerning security. It would also be interesting to know whether generic composed device templates could be made that would apply to any environment, or whether they would have to be tailored to each. It would be sensible to think that, just because a low temperature is recorded by a device that has a class of `.temperature-sensor`, it does not mean the room is necessarily cold, as it could simply reside inside a refrigerator. Whether this could be navigated by the average user would be a useful question to explore.

There are social aspects that are also worth researching (besides simply sharing device templates). It may be that users in a neighbourhood could combine aspects of their home environments to create a street-sized burglar alarm or environment monitoring station, although there would almost certainly be some trepidation and resistance surrounding that degree of sharing.

7.1.4 Physical Spaces

Experiment 2 described a system in a simple space, namely a bedroom. After this was shown to work, Experiment 4 investigated the effectiveness of a DOM-based approach in virtual representations of several other domestic and commercial environments. These also seemed to work well for the participants, as many were able to place objects and set their state, even in complex scenarios. However, these experiments only scratched the surface of what is possible in the real world. Experiment 2 showed a simple interactive environment, but a real environment would necessarily be far more complex. Even without more devices, most environments would be shared between users, rarely used for a single purpose, and would contain devices that move in and out of them.

7.1.4.1 Shared Spaces

While bedrooms and kitchens are usually fairly private spaces, many spaces are shared with either a group of known people, or else by a constantly changing stream of different individuals. At times during Experiment 2, different people used the Web UI to control the environment, although never concurrently with another user.

In both private and public spaces, some level of security would be needed to prevent users from changing states that they should not. Whether this manifests as child-friendly restrictions or simply preventing unauthorised users from affecting an environment, some level of control is required. CSS does not provide this type of write protection, as discussed later in this chapter. However, the social aspects are perhaps more interesting and less well explored than the technical issues arising.

Within a software system, multiple users will often have profiles, although in a physical space they will concurrently occupy the same environment. Shared profiles in software are often generic and purpose-driven, for instance a kiosk in a fast-food restaurant. In the IoT, it may be that a generic system is acceptable in many spaces, much as a movie theatre will have a set climate and lighting, but in other spaces, a compromise-based approach may be more desirable, even if only on shared devices. In a smart space, a control system could recognise ownership, or mediate compromises, or apply rules according to a social hierarchy.

Based on this research, research areas could manifest around:

- Combining multiple themes.
- Combining DOM trees when spaces are combined, or when new items enter an environment. This could be as simple as a laptop entering a room or opening the doors between two meeting rooms, or as complex as multiple people moving in together.
- Reconciling disagreements on state, such as room temperature.
- Applying a social hierarchy to an environment (e.g. the space owner may have more control over the environment, but a space user may have complete control over their portion of it, such as a desk space or bedroom).
- Maintaining privacy and ownership of portions of the DOM. For example, a hotel owner may have control over their building, but a guest may bring their own DOM of devices into a room; in such an eventuality, the guest would not want the hotel to control their electric blanket, but they may want to have it linked to the temperature of the room.

These problems are not limited to a DOM-based approach, but the DOM may help or hinder each uniquely.

7.1.4.2 Multi-purpose Spaces

Multi-purpose spaces are linked to shared spaces, yet are fundamentally different. They may be shared, like a staff break room, or private, as in a micro-apartment. Often a staff break room may be used for relaxation, a leaving party, a meeting, or even to sleep. Currently, staff are forced to work around the environment to make the space conducive to these activities. However, with themes designed for the space and tailored to each use, it could become as simple to change the room's purpose as pressing a button or choosing an option from a menu, as shown in Experiment 2.

Similar to shared spaces, some spaces are used for a single type of activity, yet need slight variations. During the COVID-19 pandemic, hospitals were divided into low, medium and high-risk areas – often labelled green, amber and red – [233], containing patients who had tested negative, were of unknown status, or who had tested positive, respectively. Due to the constantly shifting ratios of these groups, the colours assigned to various areas were frequently changed. Often this came with changes in use, for instance changing from an ordinary ward to a High Dependency Unit or Intensive Care Unit. These changes required a great deal of manual work to move equipment around, but this also placed a considerable mental strain on those nurses and doctors who had to remember which area was which colour. Even a simple automation to change signage and line colours between areas could have helped to ease this burden. The use of themes for this would have made such a system trivial to design and implement, provided compatible hardware was installed.

The DOM-based approach would mean that all areas of a certain use could be selected and modified at once, or else selected individually and had their designated purpose changed. Moreover, all devices and areas within those areas would also have had the changes applied automatically. Technically, this challenge would be simple to model, but the challenges would come from knowing when to implement changes, logistical challenges surrounding deployment and, perhaps most significantly, reconciling a very technical solution with a massive and diverse workforce.

Another Hospital Use Case

Another use in hospital would be to allow patients, especially those who are there for a long time, to feel more at home. At a very basic level, on entering the hospital they could choose a few simple options for colour and light temperature, along with some patterns or images they prefer. These decisions could be used to automatically build a style sheet which could then be applied to each room or bay they are moved to during their stay. It could be particularly helpful for those suffering from dementia, as a sense of permanence may help them to ground themselves. In a more advanced system, they could also define more features, such as bed heights, sounds and furniture locations.

Investigating such a system would constitute a massive undertaking, although implementing it in a way that is both digitally and medically safe would be still harder.

7.1.4.3 Dynamic Inventories

Environments may not be static. If an owner were to allow devices to be added to the DOM as they appear in an environment (e.g. a mobile phone as you arrive home), then the shape of the environment will necessarily change over time. None of the experiments however dealt with this situation, as it is tangential to the aims of this thesis, although it is likely to be a common situation within any real life IoT deployment. While there are obvious security issues attendant in allowing unknown devices to register with a system, there are many safe instances wherein environmental inventories may change significantly (e.g. the stage of a theatre).

The DOM implementation is useful here as, using the example of the stage, it could be represented as a node in the DOM whereupon only the tree beneath it would change, while areas such as seating or concessions may stay relatively static. Having a well-defined structure such as this could help in threat detection (e.g. a device joining under the stage node may be expected, but one joining the security camera node may require more investigation or monitoring).

7.2 Extensions

During the course of the experiments, several potential avenues were noted but not explored, for a variety of reasons. These are discussed in more detail below.

7.2.1 Integration with Existing Frameworks and Tools

Beginning with the most literal extension of this technology, the re-use of tools and frameworks is almost as important as the re-use of technologies, and the two are closely interlinked.

The field of Web development is crowded with frameworks and libraries that add a layer of abstraction or extra features to Web standards. The aims of this thesis all surround making a system that uses Web technologies to complement modern Web development as far as possible. Having largely succeeded in this, it would be interesting to see whether the approach could be compatible with existing Web frameworks. Given that it was shown to work with Web Components, it would almost certainly be compatible with React, Angular, and Vue (as well as others such as Polymer/Lit HTML [234]), which are the most widely used frameworks on the Web today.

Adoption by the communities of these frameworks would open up the WoT to even more developers and, as a result, even more users. Exploring the engineering changes necessary would be an interesting line of investigation, as would understanding any shifts developers would have to make to their mental models so as to accommodate it.

7.2.2 Pervasive Computing

Aspects of pervasive computing and calm technology [116], the concept that data can fade into the background of an environment, could have been implemented in Experiment 2. While that experiment reacted to data sourced from the environment as well as sunrise and sunset data, it equally could have used the same approach to react to other data sources. The closest it got to this paradigm, however, was showing the weather on a symbol-based display, although there remains much potential for further development.

The CSS-based selection would allow for the use of components of devices for displaying data unrelated to the device of which they are a part (similar to the composed devices mentioned previously). A user could have the position of an item on their desk represent whether they have an unread email or an upcoming appointment, or even the lights on their keyboard flash when they need to stand up and exercise. Data outputs could be components of devices, or else tailor-made, though generic 'data display' devices. Regardless of this, the device does not need to be designed for a specific data source or to be aware that it is being used for displaying data at all.

Much of the research here would be into implementing the technology securely and safely, while designing devices which are capable of displaying data in such a way that is acceptable and pleasing to the user.

7.2.3 Distributed DOMs

Larger deployments may require more than one hub device or DOM for many reasons, including technical limitations of the hardware and multiple ownership of spaces. Following a similar model to Experiment 2, it could be that it is more efficient to have an area represented as a DOM within a hub and that hub then shares its DOM with a more powerful remote server. The idea of a 'distributed DOM' presented itself as a possible solution to this. As the DOM is a tree, it is relatively simple to add a node above the root of two or more DOM trees which then combines them into one single tree, a process that can be done as often as necessary.

The idea of merging DOM trees has been discussed already in this chapter, but having a DOM tree spanning multiple devices has barely even been considered. Devices in Experiment 2 had the capacity to contain their own DOM as well as to register components underneath them within the hub's DOM, although this feature was not fully implemented or explored. The system also had a feature wherein messages were passed to multiple hubs and executed on each of them, but this did not treat them as a single DOM. However, messaging would be an important issue to investigate because, presumably, the root node would reside on a single primary hub, one which could easily become a bottleneck for selectors. Efficiency is also a consideration, as specific selectors may only act on a handful of devices, yet may be sent to many hubs to be executed. Such atomicity of actions may also play a part in future investigations.

7.2.4 Multiple Concurrent DOMs

Related to distributed DOMs are 'concurrent DOMs'. These arise when multiple representations of a set of devices are required. For example, a spatial hierarchy and a security risk hierarchy; in the first of these instances, the devices are arranged by their position relative to one another, but in the other, the same devices are arranged according to how much risk they present and thus grouped by the type of risk they present. This was theorised in Experiment 1, though not explored in any experiment due to the difficulty of executing it.

This concept adds a significant level of complexity to the WoT model, as selectors which work in one arrangement may not work in another, or else may have unexpected consequences. This could be solved using scoping, as is used in many Web frameworks, or could be worked around by experienced developers. However, the impact of such a layer of intricacy is unknown.

7.2.5 Media Queries

Another feature that was not implemented was that of media queries. Suggested both by Mate Marschalko [190] and by a participant in Experiment 4, this feature could allow for CSS rules to apply only in certain situations. On the Web, this is commonly used to apply styles based on device width, but in the WoT could be used for any continuous or discrete variable. For example, a developer could set a rule based on the time of day, or as an extension to the media query API, data from a sensor, or based on sunrise and sunset.

In addition to values, media queries could also be used with concepts such as ‘home’ or ‘office’, notions that would have a definitive meaning to a user, though less so to a computer system. This would be particularly useful for mobile devices, which could adapt responsively to the situation in which they find themselves.

Interesting questions develop around linking social constructs to those ideas which are sufficiently concrete for a computer to interpret. This could manifest as a system that contains a user-editable list of these social ideas, in turn linked to a set of values chosen by the user or else retrieved from an external source, potentially using the Semantic Web.

7.2.6 Abstracting Away From Classes and Properties Using the Semantic Web

The system could benefit from the Semantic Web in other ways too. While TDs have been mentioned as a template for digital twins within the system, this concept could be pushed further. As discussed later, language plays a significant role in the choice of class names and properties attached to a device. Synonyms and different languages would lead to different class names for the same concept, potentially preventing style sheets from being as effective as they could be.

One possible solution would be to abstract away from names for classes and in preference use semantic concepts. For example, rather than giving a television a class of ‘television’ the device would be given a class that is linked to the concept of a television, which can then be presented in the developer's local language and dialect. The rest of the CSS implementation would remain unchanged, though interoperability could be vastly improved.

While this is a technically sound solution, the social dimensions are far more complex. Besides forcing developers to learn the mental model for the Semantic Web, the burden of which is at odds with the core aims of this project, the Semantic Web has its own issues surrounding matching ontologies.

7.2.7 Accessibility

Another extension that was not fully explored is that of accessibility. While accessibility often seems like an afterthought to the Web, there are standards for text-to-speech interfaces and simple ways to provide text-only interfaces for those with poor sight or for conversion to braille.

Due to improved laws and generally changing opinions towards inclusivity, accessibility would have to be a part of any WoT system from the outset, although its eventual application may be very different. From a technical point of view, this may not change. For instance, a device which is a Web Component that may only need an ARIA [235] role assigned to it and the ability to read its state in text. However, the usability of such a system may be vastly different. People with different levels of sight may perceive environments differently, and the hierarchical model presented in this thesis may thus not be applicable. While visually it makes sense, the DOM tree may need to be reordered or else replaced with a proxy for users with different accessibility needs. Given the flexibility of the Web technologies it is built on, this should be possible, although rearranging the DOM would have effects on the CSS that styles it. Achieving equal access to any system may require an approach that is not present within the skill-set of Web developers or User Experience professionals. While the technology is Web-native, the accessibility approaches may not be.

7.2.8 Technology vs. Human Factors

As with accessibility, the user experience of the WoT may be vastly different for representations of physical spaces as compared to documents. It is apparent when a non-gamer picks up a controller that there is a learning curve to translating the flat visuals on the screen to a virtual 3D space. The same could also be true for WoT deployments. While Experiment 2 showed a UI controlling a physical space, and Experiment 3 showed developers learning how to control a virtual 3D space, there exists a gap between a real 3D space and the virtual representation, and this is a chasm that remains to be explored in the fields of User Experience and Perception. While it is not a Web-specific issue, it will likely have an impact on WoT systems as they reach inexperienced users.

7.3 Problems

This chapter has so far focused on the benefits and potential of a DOM-based WoT system, although it is not without problems, both technological and social. The experimental systems were all very limited in scope, yet each helped to identify issues with the approach. Some were fixed by the proceeding experiment, though others were either deliberately left, or else proved to be more foundational.

7.3.1 CSS as a Store of State

One of the most apparent issues in Experiments 2 and 3 was that CSS can represent state, though it may not be the ideal place to store it. It is inefficient to read state from a CSS document or the computed state of an element, as the entire cascade has to be resolved before the state of an element can be known. This can be optimised, as new rules and state changes can be composed with the cascade rather than having to reevaluate the whole document, although this is far from an ideal solution.

No solution to this was found in the course of this project. A backing database could be an alternative, although it would require the ability to reconcile the CSS cascade with the current state of the database, which is not trivial. Ultimately, a lot more work would need to be done to find the best resolution for this. Fortunately, despite this issue with reading state, CSS proved to be a very good solution for applying state.

7.3.2 CSS, the DOM, and Security

Another limitation of CSS is that it does not have a concept of security. There is no way to write-protect a property or even to limit who can change it. CSS on the Web has no requirement for this, as it is generally assumed that all components of the Web page are trusted and that the impact of changing the style of a Web page by a bad actor presents only a minimal risk.

A solution could be to expand the CSS specification to fit the use cases presented in this thesis. A module could be added for permissions, possibly delineating permissions using the Create, Read, Update, Delete (CRUD) approach [236], or else via an application of HTTP verbs [237]. Either would however be a further mental burden to developers, but would fit within the same mental model.

Similarly, the DOM does not have a permissions model. All scripts on the page can modify all parts of it, with the exception of some aspects of iframes and the Shadow DOM. However, a WoT DOM may have aspects that the user can change and some that it should not. Returning to the hotel example, a guest may be granted permission to alter some aspects of their room, but should not have access to read from or write to anything outside that room.

This could potentially be improved by having permissions that propagate downwards through the DOM, creating a situation where denying access to a node also denies access to its descendants. However, again this is an extension to the current system rather than remaining within its confines.

7.3.3 Physical Limitations of Devices

There is one obvious area where a Web document does not align with a physical space, in that a document contains elements that are not real in any tangible way and can be easily duplicated, moved or edited programmatically. When building Web documents, the developer usually has write access to the DOM, while the end-user has only read access. In a physical space, the layout of the environment would be largely read-only from the point of view of the developer, but the end-user would have the equivalent of write access by moving devices around and interacting with them. This could present some interesting challenges, particularly around how tools could be built that account for this in a way the developer would be satisfied with.

Other physical limitations would play a part too. For example, on a Web page, an image can be positioned anywhere but, in an environment, a device is subject to gravity and cannot overlap the space of any other device. In a fully automated environment it may be the responsibility of the tool to prevent the developer from making choices that are impossible or could cause damage, similar to how a linter works, but with far more responsibility. Finding a way to reliably present this, without placing too much responsibility on the developer, would be a significant challenge to overcome.

7.3.4 Duplication of Digital Twins

A limitation of the DOM is that each node can only appear once – i.e, a node cannot have multiple parents. This reinforces the hierarchical model, although there are some cases when it would be useful to have a node under multiple parents (e.g., the security DOM mentioned previously). In this structure, a node could be placed under a parent that determines the type of threat it is subject to. However, a device may be subject to multiple threats.

As with much of this project, there are two aspects to solving this – technical and social. The technical side provides a solution, which would be to have multiple DOMs – one for each threat type – or multiple digital twins in the same tree for a single device. The social part of the solution is in crafting the DOMs in a technically viable way, but also making sense to the humans involved. This touches upon another issue, namely domain-specific knowledge, as the arrangement of the DOMs would likely be impacted by the knowledge and mental models of the end-users.

This issue would also impact the ability to merge DOM trees, as mentioned earlier, as two trees could only merge if they did not contain the same digital twin. Although, at an architectural level, it would need to be decided whether a digital twin that is present in two DOM trees is the same twin, or else two separate twins of the same device.

7.3.5 Responses

Experiment 2 raised the problem of responses from devices. In that experiment, it was impossible to know whether all devices had acted as expected, as the total number of devices was unknown. Similarly, no response may have been an indication of no devices acting on the request or simply there being no devices that chose to respond. There was also the potential situation in which no devices were currently online, yet they could have acted on it in the future when they were registered and synchronised with the state of the hub's internal style sheet.

The solution is not immediately apparent as it stems from the zero-knowledge aspect of the system, one which enables several of the aforementioned benefits. Hopefully, it could be overcome, or perhaps the initiating device does not need to receive a response, much like UDP [238].

7.3.6 Languages

The early Web was very Anglo-centric, having begun life in the USA and branching out first to the UK. This legacy has lived on and, today, all specifications and much of the day-to-day business of the Web occurs in English. However, English is not the first language of most of the world. When building the Web, this was not a large consideration to developers as, with the implementation of Unicode, they were free to choose variable names and write comments in their own language (except in those instances in which they had to interact with libraries, external resources, and APIs). CSS is one such API – the properties are all English – and so far little or no attempt has been made to change this. Class names, however, can be in any language, or use any Unicode string (e.g emoji class names are perfectly valid in both HTML and CSS).

The same is not true in a WoT system akin to those shown in this thesis. A deployment could contain devices from many manufacturers from across the world and, even if they chose a common language for implementation, bad translations and spelling mistakes would disturb interoperability considerably. Even within English, many synonyms could be used by different people. This could be worked around using the Semantic Web integration as mentioned previously, or via the use of a closed taxonomy of property and class names, similar to CSS's closed list of properties.

7.3.7 Open vs Closed Property Taxonomies

The current CSS specification contains a closed taxonomy of properties and, for the most part, the values of those properties are also restricted. Experiment 2 used a closed taxonomy, while Experiments 3 and 4 had an open taxonomy using Custom Properties. Neither presented an immediate issue, as all the devices were created by the same person and control over them was absolute. Both alternatives have benefits and problems and consequently neither embodies an ideal solution.

A closed taxonomy has the potential to never be sufficient for every use case, leading to misuse of properties in cases in which there is no perfect choice. It could also grow to an extremely long list that is hard for a developer to keep track of. Even with an exhaustive list, it is still open to misinterpretation where words have ambiguous meaning.

An open taxonomy could leave the system susceptible to developers creating multiple properties that have the same meaning, either in the same language or different ones. It could also make generic style sheets almost impossible to produce and maintain, as the author would not know which properties to set to achieve their goals across multiple devices.

The Semantic Web could provide a partial solution, as mentioned earlier. It could link properties to meanings and thereby allow them to be language agnostic. It would also facilitate a structured, semi-open taxonomy in which each manufacturer or developer uses their own vocabulary which is codified in a shareable document linked to the style sheet.

In conjunction with this, or alternatively, smart tools could inspect an environment, similar to the inspector present in the developer tools of most Web browsers. This could give developers a list of available device types and properties, enabling them to understand the environment slightly better.

However, either of these would again be an extension of the Web standards and associated models, thereby placing an increased burden on the developers using the system.

7.4 Final Thoughts

Despite these issues, it is hopefully clear that the use of the DOM to model an IoT environment presents not only interesting and expansive research opportunities, but also a chance to make the WoT more widely accessible to Web developers than any other existing approach.

As has been shown throughout this thesis, it is possible, practical and acceptable to the community to produce a WoT system that abides by the constraints, rules and intentions of existing Web technologies. The final system (presented in Experiment 3) is very close to a complete control system and, coupled with a back end that could communicate with real devices similar to that used in Experiment 2, it could prove a fairly rounded prototype for a commercial system. From an academic perspective, the project has been very successful. However beyond the realm of the hypothetical, as with most concepts, it would take a lot of work, belief and dedication to progress the idea beyond the laboratory.

Many of the ideas presented here surround the direct implanting of devices into the DOM and then using CSS to apply state. These concepts are almost entirely unique, especially in terms of their scope and depth. The related implementations by Martin Schuhfuß [1] and Garza et al. [2] are far more superficial with very different aims, and Mate Marschalko's [190] ideas came later and are far less developed than those in this project. Martin Schuhfuß' demonstration using CSS to control lights was intended for the stage and as a curiosity at a conference. While it used some of the benefits of CSS, such as classes, it did not use a DOM, a cascade, or any other browser technologies. Instead, it simply converted CSS into DMX [189] commands and sent them to the lights. Garza et al.'s approach to using CSS and the DOM to build IoT devices was tangential to their aims and the inverse of what is demonstrated here. Where they mapped the virtual to the physical, here the physical is mapped to the virtual. They used a DOM and CSS, although as a definition language for a device, rather than to mirror the state of an existing device. Both are interrelated, but fundamentally different, and neither comes close to a full implementation. It is worth emphasising here that all three were instances of parallel thought alongside this thesis and one another and so begin with different problems and end with different solutions.

The experiments within this thesis show the start of what would likely be a much longer journey to a system that fits perfectly with the Web, not least because standards such as those from the W3C WoT Group are yet to be finalised. The code used to compose the experiments in earlier chapters was state-of-the-art for the Web, but the Web develops very quickly. The W3C is not the only group to be actively working on the WoT, though they are the foremost towards achieving a complete solution. The Matter Group is closest on the commercial side, but has yet to release much about how they plan to tackle the WoT.

Experiment 2 is a prime example of the rate of obsolescence in this area. While at the time it was made it used the most up to date APIs offered by the browser, it has very quickly been superseded by the Custom Properties and Web Components APIs which could replace much of the code within the hub with standardised, streamlined, faster, and more reliable alternatives. Experiment 3 demonstrated these, although it too will become obsolete very soon. However, both played important developmental roles. Experiment 2 showed that an efficient and fairly stable WoT system could be built based on the DOM, CSS and JavaScript, a feat that was essential to this thesis. Experiment 3 was as forward-looking as it is possible to be, and its simulated nature allowed for the imagining of devices beyond those actually possible today. It looked forward to a world in which almost every device is connected to and a part of the IoT environment and, moreover, successfully attempted to provide a familiar interface to that environment. It demonstrated that, not only would developers be able to use such a system, but also that they could adapt to both the interface and mental model remarkably quickly.

The impact of the technology would be incremental to the evolution of the Web, but the benefits of employing a system that is straightforward to use and develop for are fairly considerable. A familiar overlay for a complex problem could be the difference between expanding an existing domain of knowledge and creating a new one. Familiarity would hopefully enable both a faster adoption of the WoT along with a lower mental burden to already taxed developers. With this could come lower training costs and a larger pool of developers to pull resources from which, together, would lower both costs and the barrier to entry. On a personal level, it would require developers to be less specialised and allow them more flexibility in their career choices. They would not have to make a conscious decision to become either a Web or a WoT developer, as they could happily be both due to a common mental model.

The approach here may initially appear to be another competing standard. However, it could be framed as a complementary solution, one that augments existing and future WoT proposals. It is already theoretically compatible with what is known about Matter and the W3C's WoT implementation and, as these seem to be the dominant players in the WoT field, this is a good position, although it is still very early in the game. This compatibility is key, both to the aim of the thesis and to its potential for adoption. It is hoped that working within the existing landscape in this way would result in a far better result for the community than a more radical alternative.

Appendix A

Experiment 2: Message Schemas

Key:

`r:`	`message ID`
`o:`	`originating device ID`
`s:`	`selector`
`t:`	`type`
`st:`	`status`
`m:`	`message`
`d:`	`data`
`dv:`	`device`
`ch:`	`channel`

Response failure reasons:

`0:`	`ok`
`1:`	`general failure`
`2:`	`timeout`
`3:`	`specific failure, see message`

Create a device in the DOM:

```
{
  "create": {
    "r": "12345",
    "o": "some_device",
    "dv": {
      "id": "ceiling_light",
      "class": "light ceiling",
      "attr": {
        ...
      }
    }
  }
}
```

Create a component under a device in the DOM:

```
{
  "create": {
    "r": "12345",
    "o": "some_device",
    "cm": {
      "id": "ceiling_light",
      "class": "light ceiling",
            "p": "parent",
      "attr": {
        ...
      }
    }
  }
}
```

Create a channel in the DOM:

```
{
  "create": {
    "r": "12345",
    "o": "some_device",
    "ch": {
      "id": "ceiling_light",
      "class": "light ceiling",
      "attr": {
        ...
      }
    }
  }
}
```

Return the DOM element for the device:

```
{
  "read": {
    "r": "12345",
    "o": "ceiling_light",
    "s": "#home.bedroom#light",
    "dv": {}
  }
}
```

Returns the DOM element for the channel:

```
{
  "read": {
    "r": "12345",
    "o": "ceiling_light",
    "s": "#home.bedroom#light",
    "ch": {}
  }
}
```

Query a device for it's representation:

```
{
  "read": {
    "r": "12345",
    "o": "ceiling_light",
    "t": "discover"
  }
}
```

Update the device representation in the DOM:

```
{
  "update": {
    "r": "12345",
    "o": "ceiling_light",
    "s": "#home.bedroom #light",
    "dv": {
      "class": "multicolor touch",        // replace, or
      "class": "+multicolor touch",       // add, or
      "class": "-multicolor touch",       // remove
      "css": {
        "background-color": "burgandy",
        "power": 1
      }
      "d": {
          ...                             // send some data
      }
    }
  }
}
```

Update a device's id/class (on the device itself):

```
{
  "update": {
    "r": "12345",
    "o": "ceiling_light",
    "s": "#home.bedroom #light",
    "dv": {
      "id": "new_id",
      "class": "multicolor touch",        // replace, or
      "class": "+multicolor touch",       // add, or
      "class": "-multicolor touch",       // remove
    }
  }
}
```

Update a channel in the DOM:

```
{
  "update": {
    "r": "12345",
    "o": "ceiling_light",
    "s": "#home #temperature",
    "ch": {
      "subscribers": "#some_device",      // replace, or
      "subscribers": "+#some_device",     // add, or
      "subscribers": "-#some_device",     // remove
      "class": "temperature",             // replace, or
      "class": "+temperature",            // add, or
      "class": "-temperature",            // remove
      "d": {
          ...                             // send some data
      }
    }
  }
}
```

Remove a device from the DOM:

```
{
  "delete": {
    "r": "12345",
    "o": "some_device",
    "s": "#home.bedroom#light",
    "dv": {}
  }
}
```

Remove a channel from the DOM:

```
{
  "delete": {
    "r": "12345",
    "o": "some_device",
    "s": "#home.bedroom#light",
    "ch": {}
  }
}
```

A response from a device after it has acted on a message:

```
{
  response: {
    "r": "12345",
    "o": "some_device",
    "s": "#some_device",
    "t": "update",                        // type: discover, create, update, delete
    "st": 1,                              // status: 1: success, 0: failure
    "m": "",                              // message: optional
    "dv": {},                             // device: optional
    "d": {                                // data: optional
      ...
    }
  }
}
```

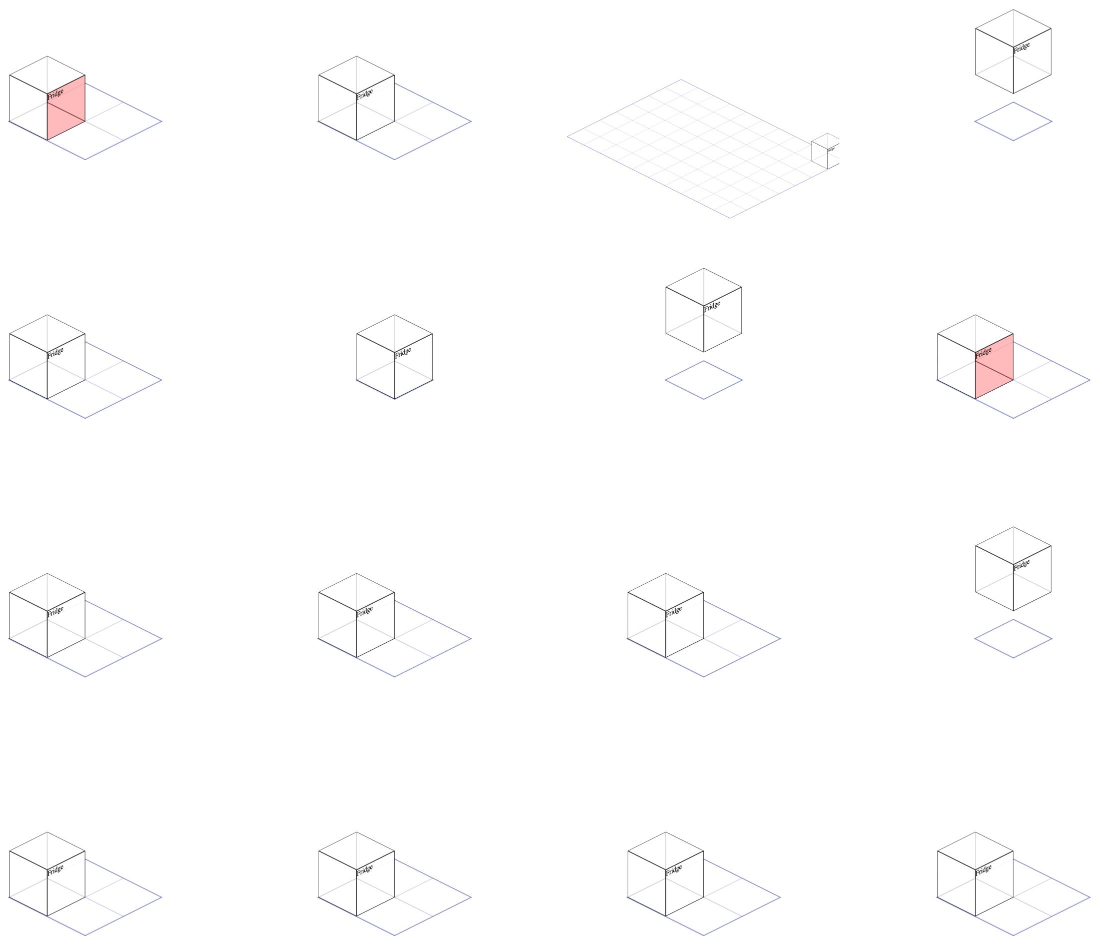
Publish to a channel:

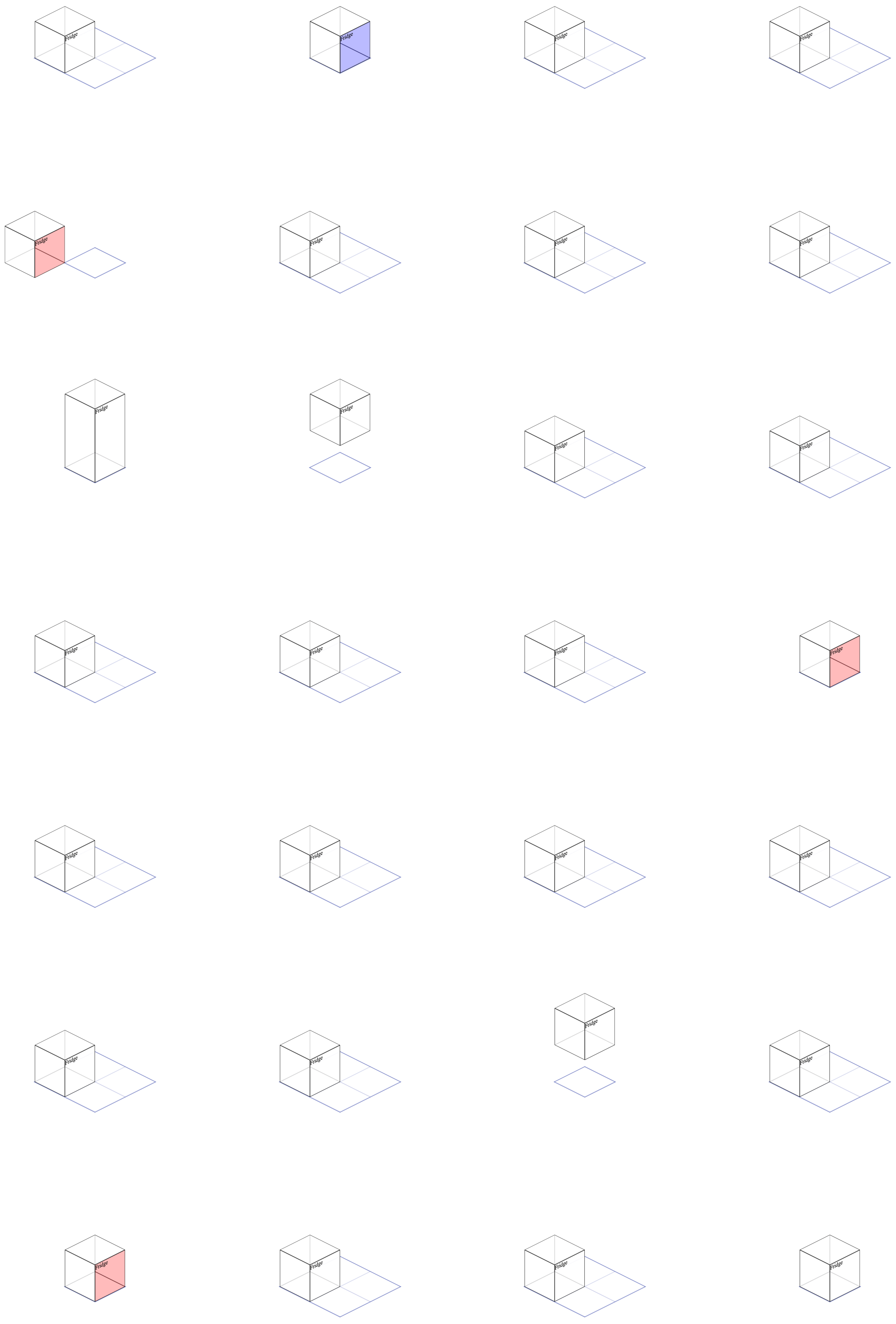
```
{
  "publish": {
    "r": "12345",
    "o": "ceiling_light",
    "s": "#home.temperature",
    "d": {
      ...
    }
  }
}
```

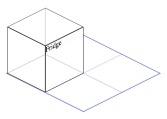
Appendix B

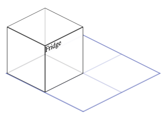
Experiment 4: Participant Scenes

Part 1

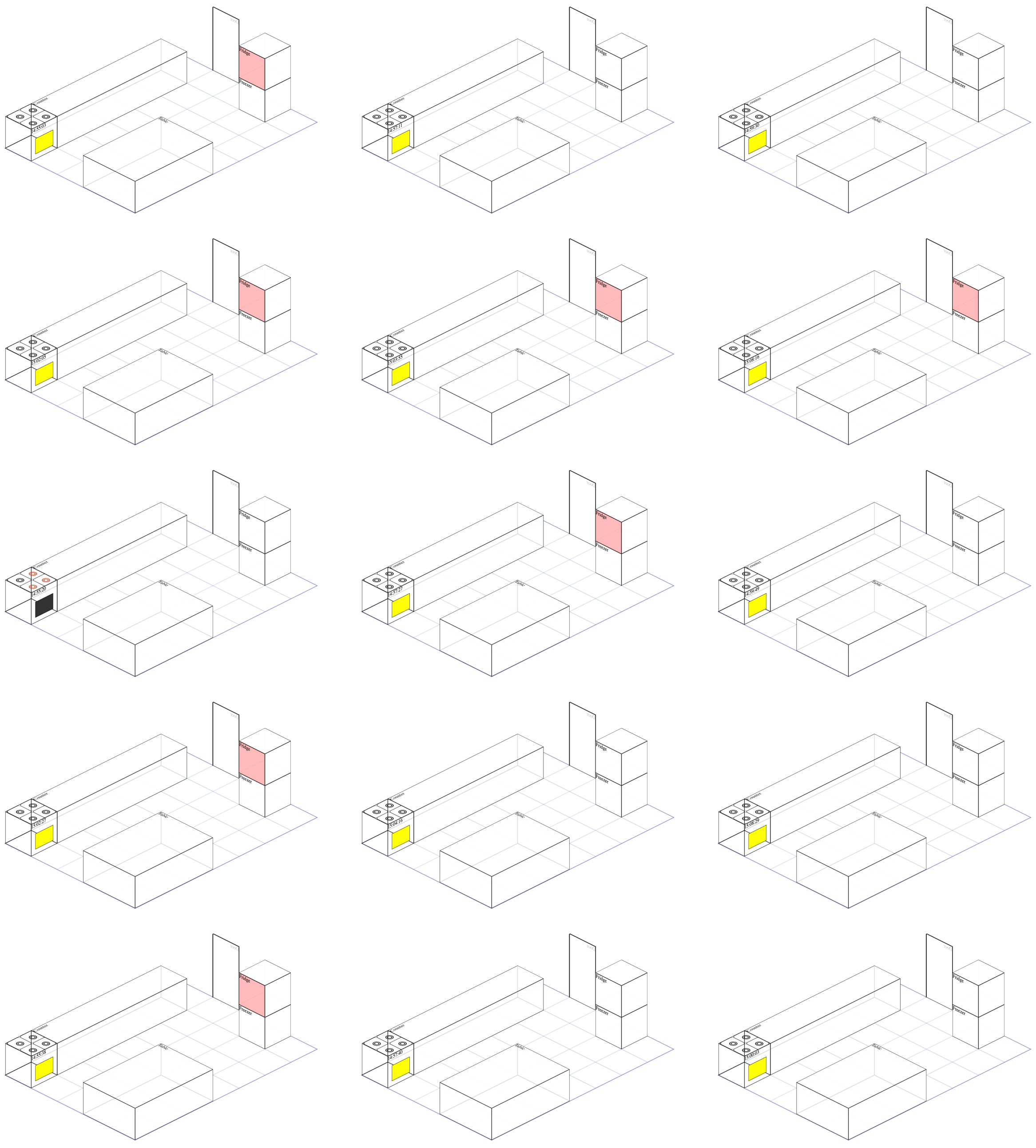


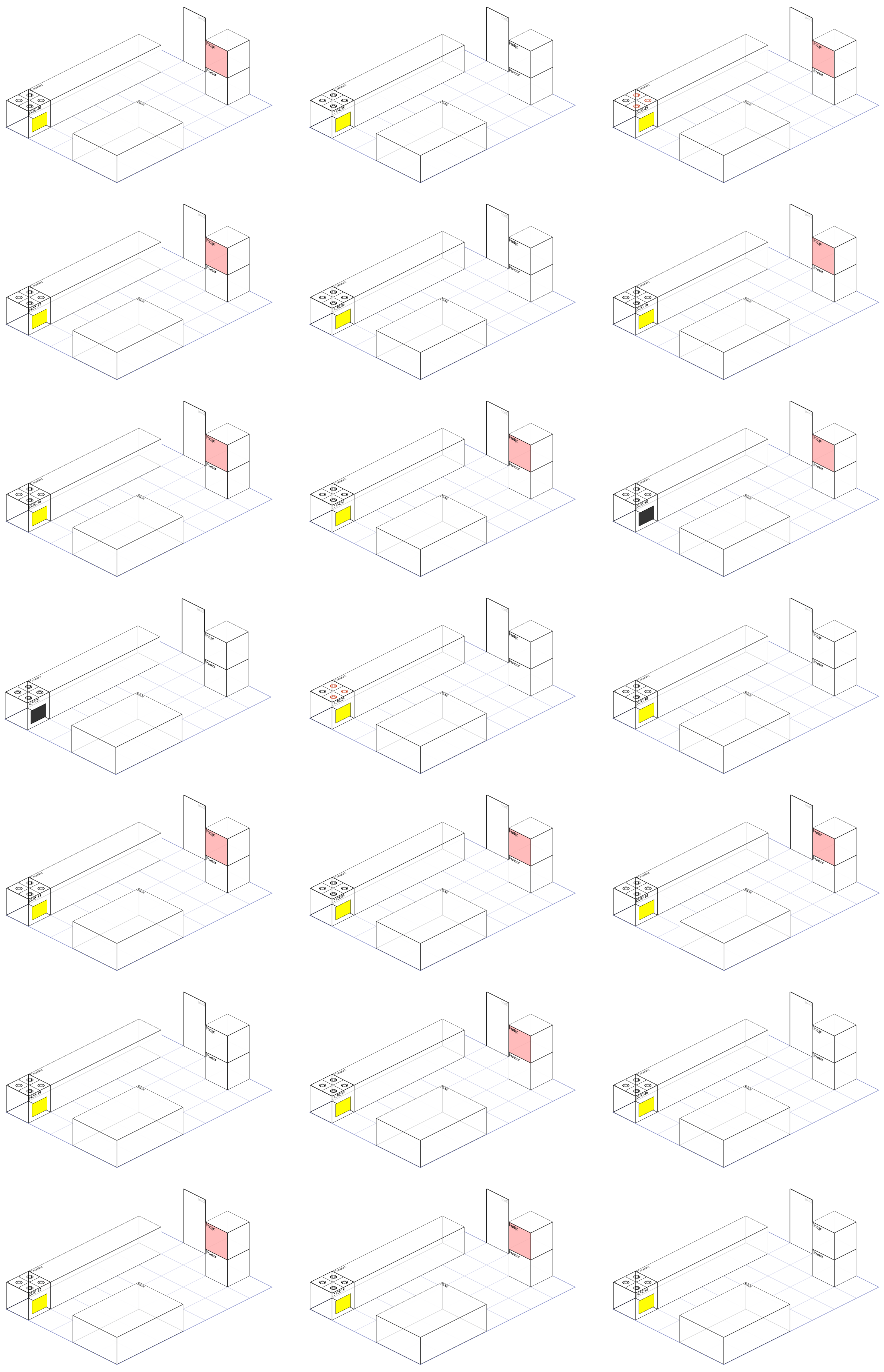

Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge
Fridge

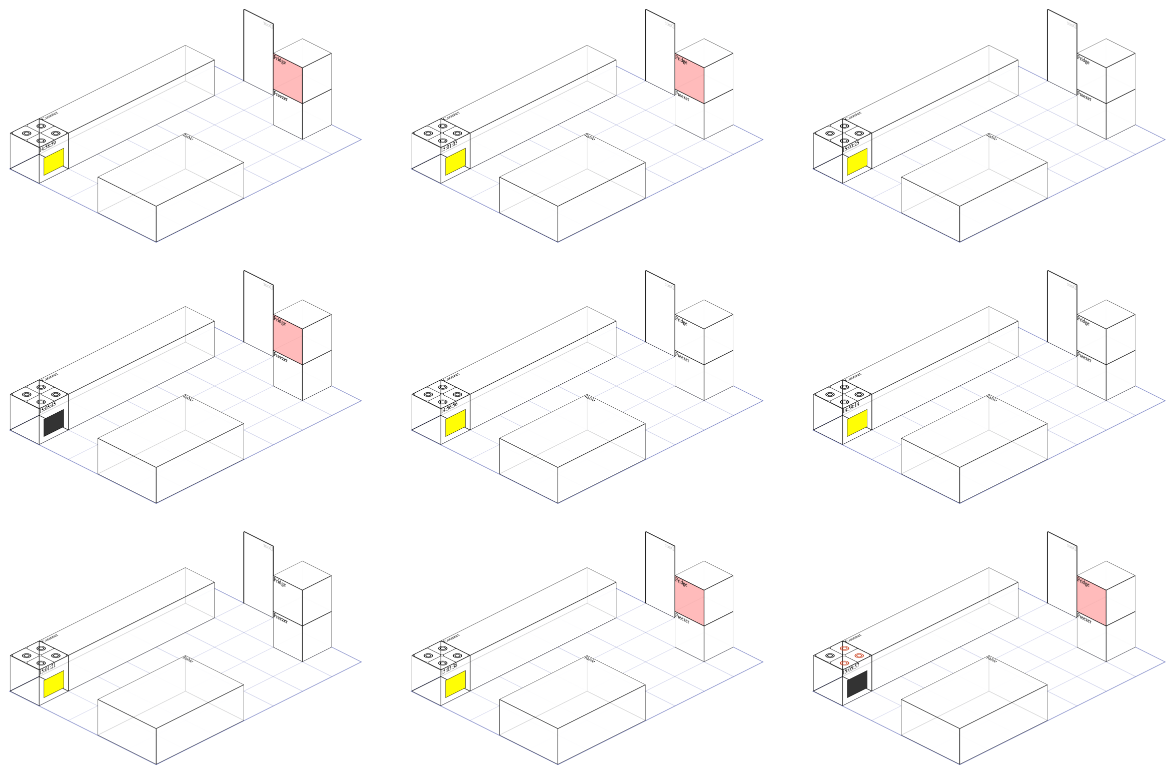




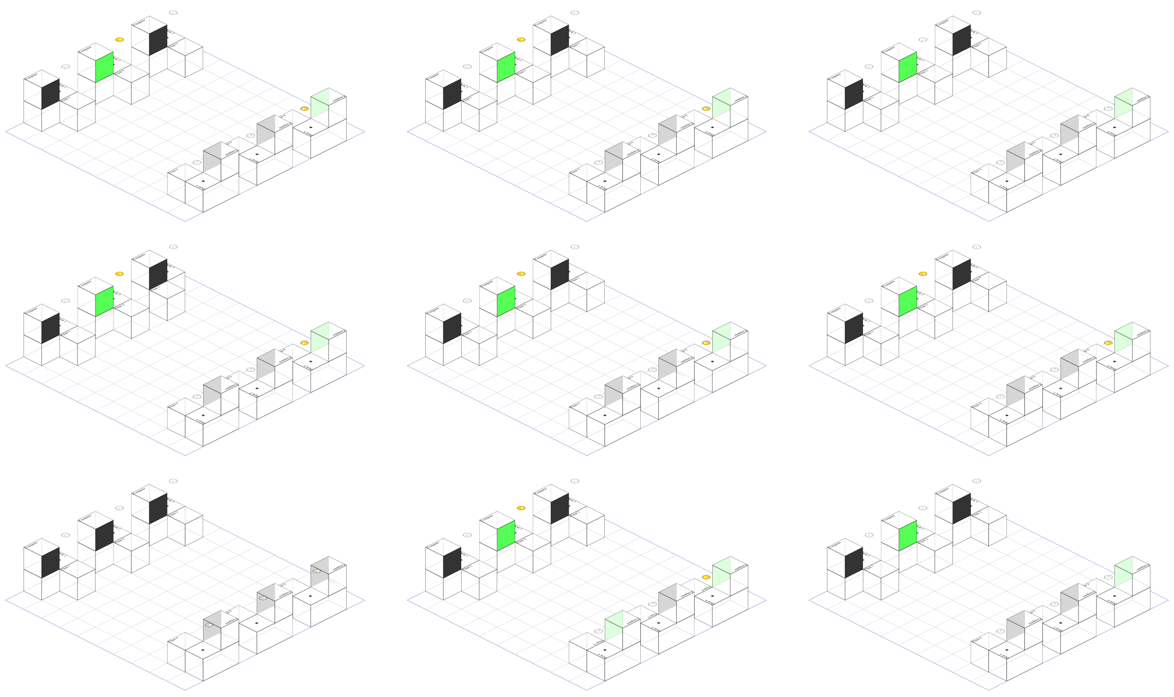
Part 2

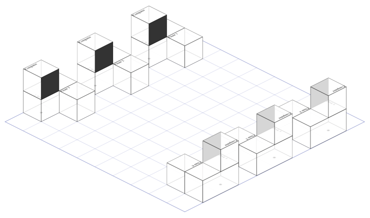
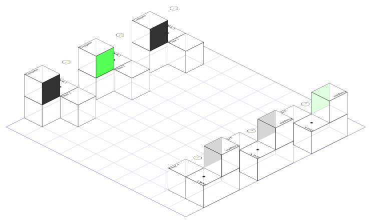
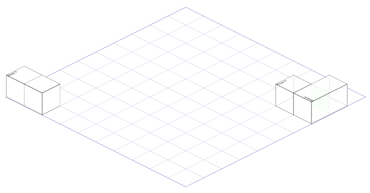
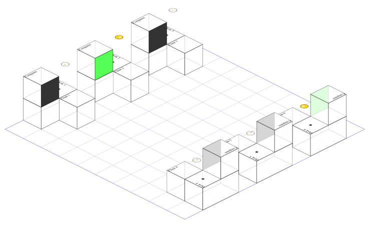
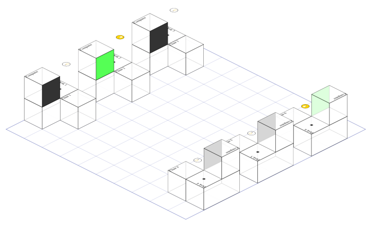
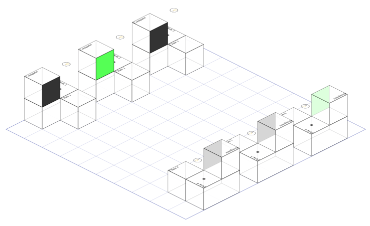
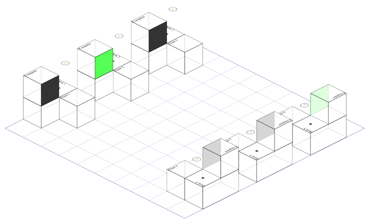
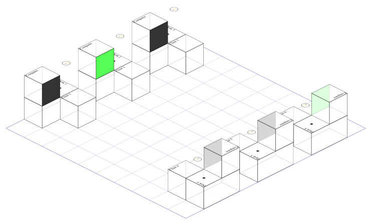
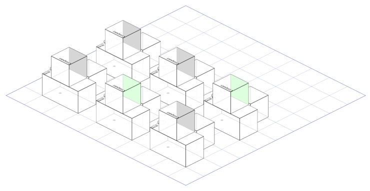
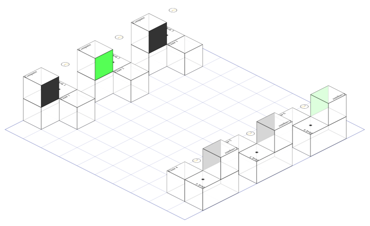
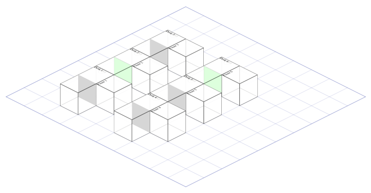
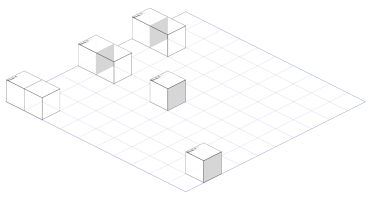
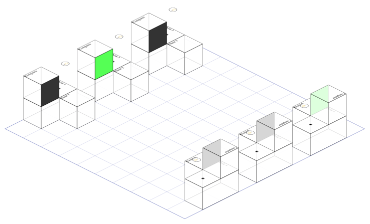
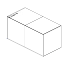
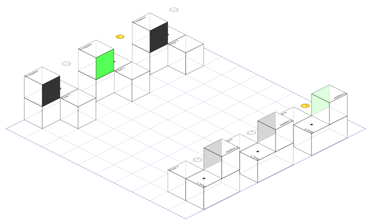
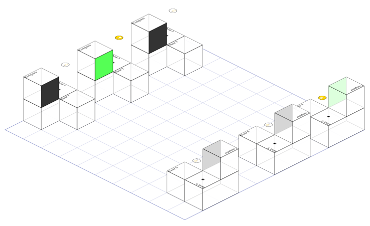
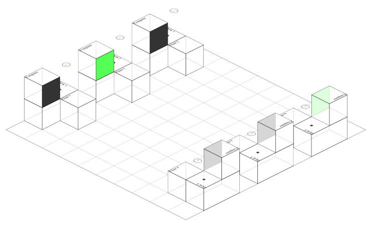
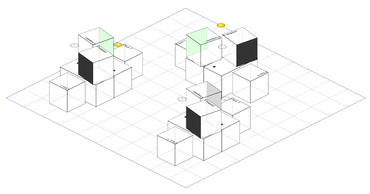
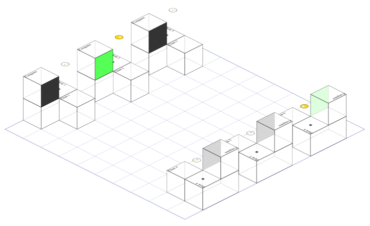
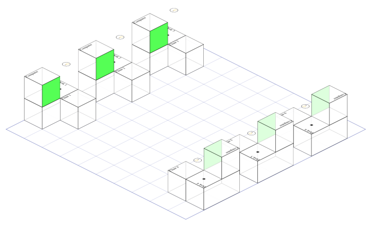
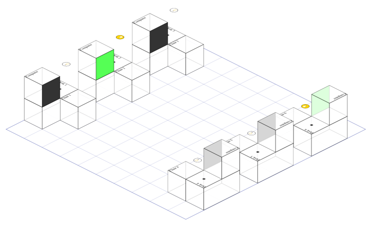


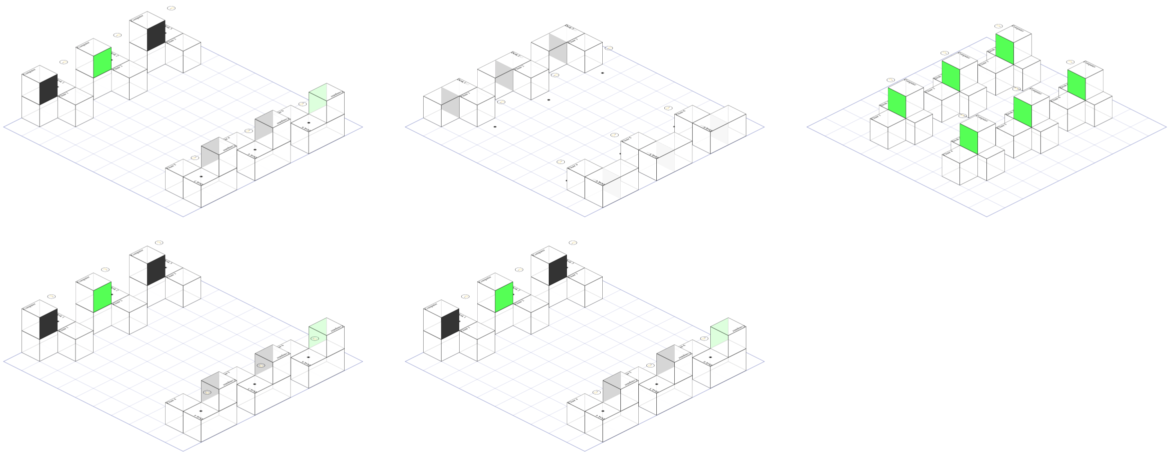




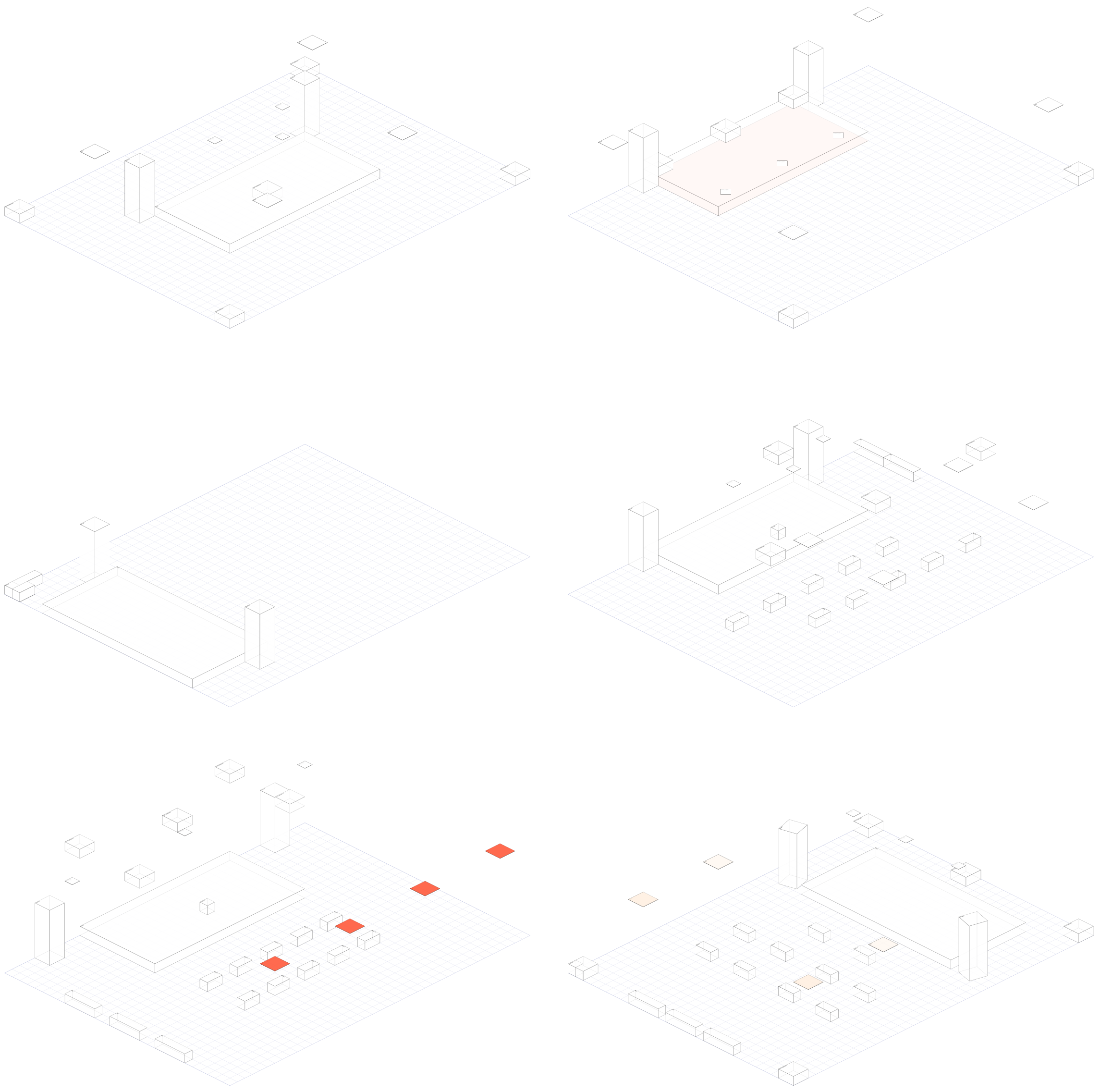
Part 3

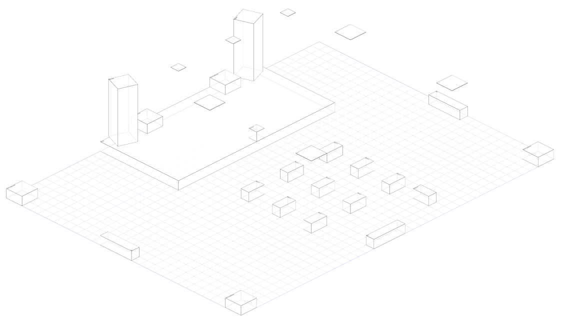
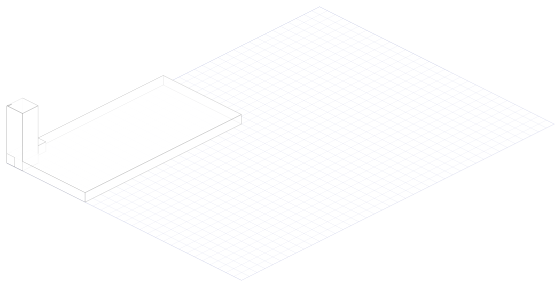
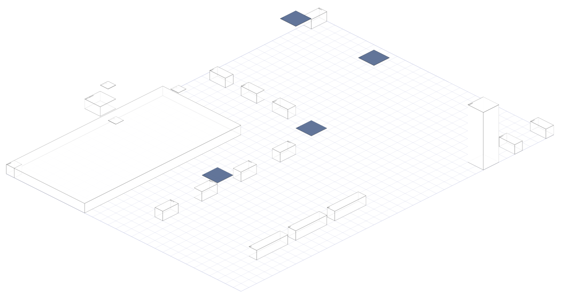


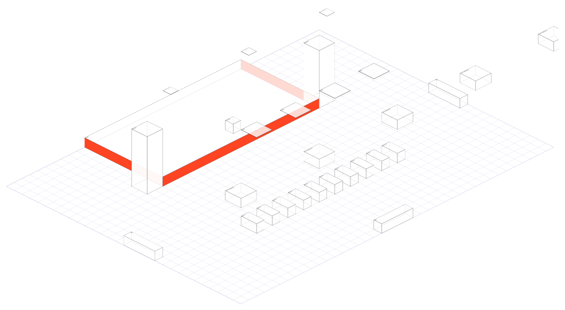
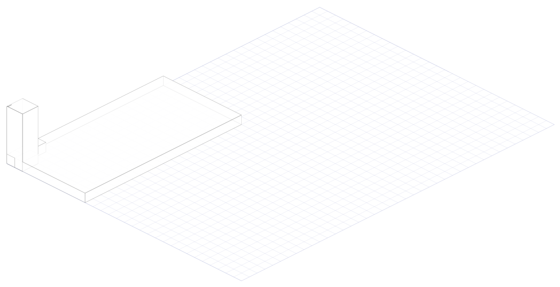
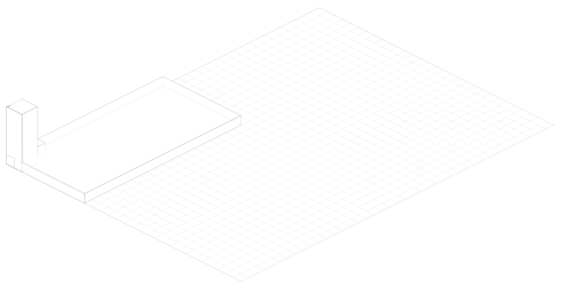
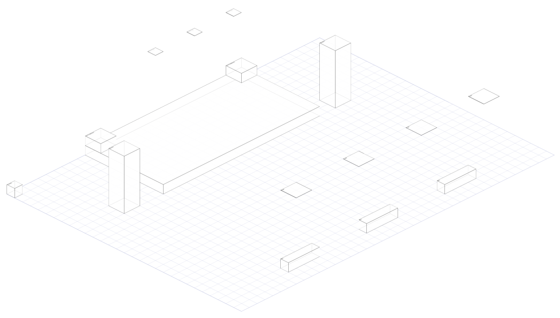






















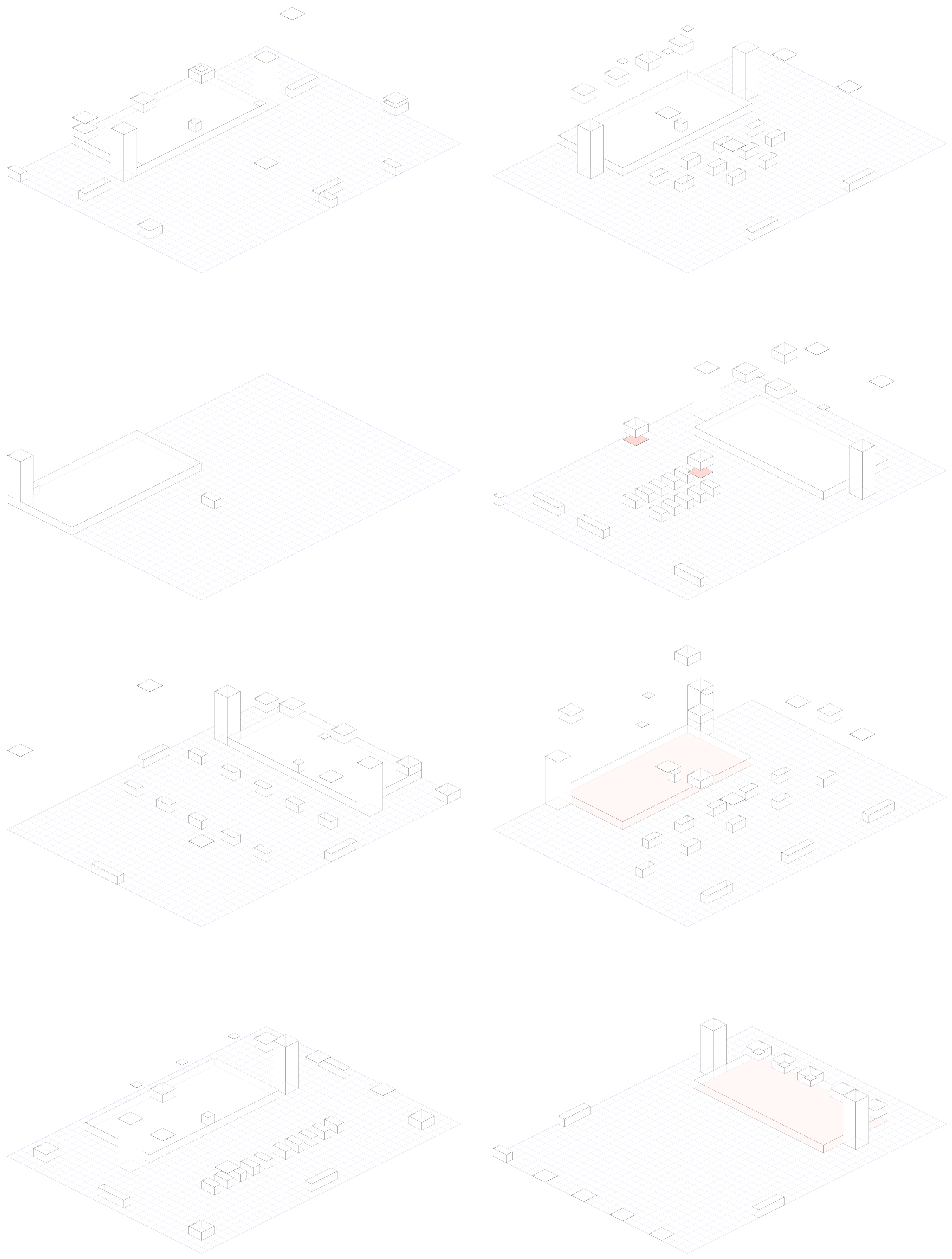


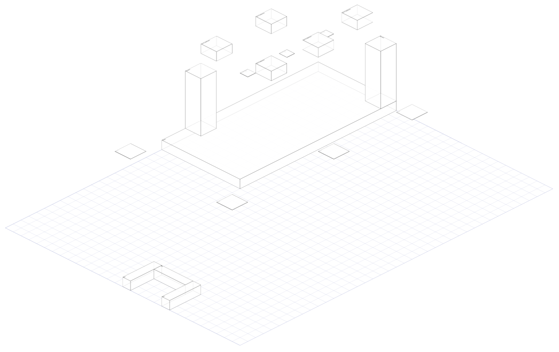
Part 4

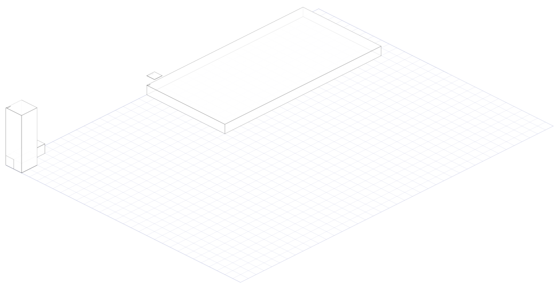


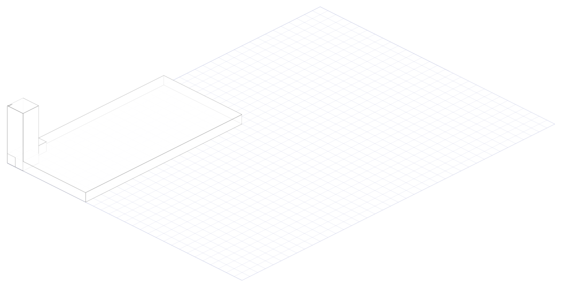












Appendix C

Experiment 4: Participant Scene Descriptions

ID 60d87ec5b26cea00154d3111

Before

The Image shows a room with an aspect ration of around 2:1. I will describe the position of things in the room using the compass. The north-south side is the longer one. The door is in the middle of the northern wall, east of it is a lit-up shoe rack with a painting above it, and theres a window on the west wall. next to it is a bed with underglow. In the south-east corner is a gaming setup with an angled table and a chair. There is an overhead fan-light. In the north-west corner there is a cupboard with magazines in it. Behind the headrest of the bed are multiple lights.

After -

ID 60dcb0b9fbb87600156f90a6

Before

A square bedroom rotated at 45 degrees. It contains an L shaped desk in the bottom corner with the opening to the bottom left wall. On the desk is a computer with four monitors and a laptop. There is a gaming chair in the opening of the desk. There is a bed in the left corner with night stands on either side of the headboard. There are orange lights coming from the bottom of the night stands. There is a hardwood floor. There is a bookshelf in the top corner with two upper shelves and a lower cabinet. On top of the bookshelf are some boxes. There is a window in the middle of the top left wall with a poster of Darth Vader to the left of it. There is a door in the middle of the top right wall with a poster to the right and two smaller posters to the left. There is a ceiling fan in the middle of the room.

After -

ID 60dcbd30fbb87600156f90f6

Before

Room is rectangular. Enter at long side of the room, at one third – shorter path of the wall on the right.

Starting from your left, going clockwise through the room:

- shoo cabinet with 5 boards, right next to door.
- above shoo cabinat a wallpaper / painting
- first corner
- IN the second corder: desk with pc, laptop, triple monitor, headset holder and 2 speakers, keyboard, mouse, mousemat. L-shape desk. User would sit such that it is facing first corner. Gaming chair + chair mat underneath.
- right before third corner, bed with back to wall. Lamp + case at both sides of bed.
- third corner.
- poster
- window
- fourth corner.
- bookcase
- door.

After -

ID 60dcd49dfbb87600156f924d

Before

An isographic rendering of a bedroom, which is rectangular. The room is longer than it is wide. The room's floor is a dark purple type of tiling. An L-shaped desk sits closest to the viewer, with a PC tower on the closest side, three monitors in the corner, and a laptop on the furthest side. The outside corner of the L is facing the right. A gaming-style chair sits inside the L on top of a mat. To the left of the desk is a bed with two side tables on either side, with the bed facing the door on the other side of the room. Both side tables have lamps with rectangular prism shades. A window & shades are on the middle of the wall far to the left. To the left of the window is a dark-colored poster. In the far corner of the room is a bookshelf with two shelves of books or games, one shelf with other items, and the top shelf with three cubic items. The door is opposite the bed on the far right wall fo the room. A poster or possibly a TV hangs to the right of the door. Below this item is a shoe rack with five shelves and with three pairs of shoes, two on the lowest shelf and one on the second-to-lowest shelf. A ceiling lamp & fan hangs slightly over the space between the bed and the desk.

After -

ID 60dcebc7fbb87600156f956e

Before

This is a bedroom. On the ground, there is a bed, 2 night stands with lamps on them, bookshelf, shoe rack, l-shaped desk with chair. On the walls, there is a window to the East side of the room, and a wide poster on the North side of the room. The bookshelf height extends to almost the ceiling, while everything else extends to abound waist height.

After -

ID 60dcf044fbb87600156f9588

Before

```
 {
   lengthX: 8,
   lengthY: 13,
   lengthZ: 8,
   objects: [
      {
          object: "bed",
          offsetZ: 0,
          offsetX: 0,
          offsetY: 1,
          orientation: -90
      },
      {
          object: "nightstand",
          offsetZ: 0,
          offsetX: 0,
          offsetY: 1,
          orientation: -90
      },
      {
          object: "desk",
          offsetZ: 0,
          offsetX: 0,
          offsetY: 13,
          orientation: 90
      },
      {
          object: "ceiling-fan",
          offsetZ: 8,
          offsetX: 2,
          offsetY: 5,
      },

   ]
}
```

After -

ID 60dd2e0e84bb8c0015a4a1a5

Before

The setting is a medium sized bedroom around 12 units long by 7 units deep.

Along the back wall, from left to right, are these items:

1. A bookshelf that has drawers, approximately two units wide, two units deep, and reaches up to 1.5 units below the ceiling. The bookshelf is populated by several books on the top two shelves, an miscellaneous items on the bottom shelf. On top of the bookshelf are 3 cube-like structures.
2. To the right of the bookshelf, two paintings on the wall, oriented vertically from one another.
3. To the right of paintings, a white (closed) doorway that is approximately 1.8 units wide, and reaches about the same height as the bookshelf.
4. To the right of the doorway is a shoe rack that is approximately one unit wide, above it is a wide painting of an aurora borealis.

Along the left side of the room is a painting of an alien creature, and a closed window around 3 units wide.

Along the front side of the room, there is a queen-sized bed with two modern nightstands, each with a lamp on top. Then to the right of the bed is a gaming setup, composing of an L-shaped desk with triple monitors, a laptop on the left, a racing chair, and a desktop on the right.

After -

ID 60dd2fa584bb8c0015a4a1a6

Before

Isometric view of a room

Top corner is a bookshelf with books and some things on top.

Right top wall has the bookshelf, the door, an aurora painting and a shoe rack.

top left wall has a whiteboard in the center, a darth vader poster and a bedside table with a lamp.

Following along to the lower left wall of the room there is a bed, a second bedside table with a lamp and a desk with a computer and 3 monitors.

After -

ID 60dd5e6684bb8c0015a4a2e3

Before

Viewing from the door there is a window with blinds in front in the right wall, against the wall the door is on there is a bookcase on the right and a rack with shoes on the left. Above the rack is a painting of the northern lights. in front of the door there is a twin sized bed with on both sides tables with lamps on them. to the left of the bed is a L-shaped desk with three monitors and a laptop on it, next to the desk is a racing chair. underneath the desk and chair lies a black rug. from the ceiling hangs a fan with light.

The floor is made up of brown tiles.

After -

ID 60dd628a84bb8c0015a4a323

Before

(from the perspective of the door) There's a window on the center of the right wall, with a poster to the left of it.

Right after entering the room, there's a cabinet and shelving on the corner. Right in front of the door there's a queen-sized bed with small bedside tables on both sides, with a lamp on top of each one.

In the center of the room there's a ceiling fan with the rooms light.

On the left there's a small shoes shelve and on the left corner, to the side of the bed, there's a computer desk that's L-shapped to the wall, with a "gaming" chair on it. There's three screens on the bend of the table, a keyboard and a macbook to the left (from the sitting position) and the computer is to the right.

After -

ID 60dd72bc84bb8c0015a4a453

Before

The room is a rectangular room, with the long sides around 1.5 times as long as the short side. Slightly to the left of the middle of the long wall opposite the camera is the door. Left of this, in the corner, is a cupboard with the same width as the door. Right of the door is a shoe rack. In the corner closest to the camera is an L-shaped desk with the chair inside the L-shape against the other long wall. Against this same wall, more to the left, is a 2-persons bed, with small nightstands at both sides of the headrest. On the left short wall is a whiteboard/window. In the middle of the ceiling hangs a ceiling fan.

After -

ID 60dd78cf84bb8c0015a4a4a0

Before

This is an isometric rendering of a gamer's bedroom, at a 45 degree angle as is common for isometric renderings. Let's say the NW to SE axis is our x axis and represents "from left to right" and the NE to SW axis is our Y axis. The left and top wall are drawn as to allow us to look into the room. In the bottom left is a king size bed, in the bottom right is a desk with a gaming chair and three monitors, in the top left is a bookcase. Most elements have some type of LED light illuminating the side.

After -

ID 60dd89baa3ae320015ef2226

Before

An isometric view of a bedroom.

The grid of the room is 7x12x5 (x, y, z) squares. Each square is approx $0.5m^2$.

The point farthest from the camera view is defined as 0,0,0.

The two closest walls and ceiling are not rendered.

Objects in the room (corners):

0,0,0 - 1,0,3 = Shelf with draws
3,0,0 - 5,0,4 = Door
6,0,0 - 7,0,2 = Shoe rack
1,3,0 - 4,7,2 = Bed
0,7,0 - 0,7,2 = Bedside table
5,7,0 - 5,7,2 = Bedside table
7,3,0 - 12,7,2 = Corner computer table with chair

Note: If I had longer I would put more detail on object positioning

After -

ID 60dd9045a3ae320015ef223d

Before

A bedroom of a possible streamer, who likes to stream games. He has a high-tech setup with 3 monitors next to each other, a big tower rig, sound-boxes, a fan and a macbook. A Gaming-Chair on a carpet. Lots of colorful lights everywhere. There's a shoe rack that has blue light behind it, above that is a big TV with a very wide angle. The bed as well as the nightstands beside it also have colorful light behind it. Theres a Poster of Darth Vader and a Window (which is closed) next to it. He has a big collection of games on a shelve. No plants though, nothing green. Very clean though.

After -

ID 60dd991da3ae320015ef22dc

Before

A bedroom about 6m x 3m x 2m. A white door to enter the room on the lengthwise of the room, about half a meter wide, with a shoe rack (and a poster above it) to the left of it, with more, smaller posters to the right of it, and more to the right is a bookcase with gadgets on it. A window posts up on the widthwise to the right of the door, with a poster on the left of it. 2 lamp stands with a lamp on each sandwich a king sized bed with mood lighting on the opposite lengthwise wall that is pressed to the wall with the window. A computer, a laptop, a triple screen monitor setup with wireless keyboard, mouse, and headset, is present on a table that is shaped like an L, pressed to the other widthwise wall and the user will be sat facing the door. Finally, a gaming chair and a mat that prevents scratches on the floor can be seen.

After -

ID 60dda539a3ae320015ef22f9

Before

Reference frame: standing in front of a bed turned towards it. The size (Or rather the ratio) of the room is around 1:2 (Width:Length). A bed is located in front of you, it is a double person one. On both sides on the bed there are 2 boxes/tables with 1 lamp on each. Right behind you is a door, and on the right of the door there is a bookshelf and on the left of the door there is a shoe holder. On the right wall there is a table that is roughly "L" shaped with a computer, 3 monitors and a laptop and a gaming chair right beside it. There is a ceiling fan with a lamp mounted in the space between the bed and the table.

After -

ID 60dde5e4a3ae320015ef23dd

Before

It is an isometric render of a bedroom. Two walls are visible, the other two and the ceiling have been eliminated so that the inside of the bedroom can be seen. The floor is made of shiny brown tiles. There is a window in the “top left” wall and the blinds seem to be down. That same wall has a poster. On the “top right” wall there’s a wooden bookshelf, right on the corner between the two walls, facing the “bottom left” wall. There are two posters right next to the bookshelf, and then there is a white door which is closed. Next to the door, there is a wide painting of what looks like the Northern Lights, and below it, a shoe organizer with 3 different pairs of shoes, that glows for some reason.

The wall that’s closest to the camera (“bottom left”) is where most of the interesting action is happening. From left to right, we can see a bedside table with a small lamp on it, then a big slightly undone bed, then another identical bedside table with another lamp.

In the corner, there is a computer desk that has a laptop, a 3-monitor setup, a keyboard and mouse, what look like speakers, and a desktop computer tower. The desk is in the shape of a horizontally flipped L, with the long part of the L “pointing” towards the bed. In the inner space, there is a gaming chair.

After -

ID 60ddee96a3ae320015ef247b

Before

My room starts with a small corridor that is about 1m long, after which you enter the room. In front of it, there is a small closet. On the left side, there is a Big Closet. On the left wall, there is my desk with my computer. On the right side of my desk, there is my bed. Vis-a-vis of my bed, there is a couch.

After -

ID 60e0ad89ac638800156ed021

Before

```
Var ceilingLightArray;
ceilingLightArray(
    X:2;
    Y:2;
    Z:3;
    Type:spot;
    Colour:ffffffff;
);
```

After -

ID 60e1aeaf553e3500157a87ef

Before

```
/*
 5 mins is far too short a time for me to describe something like this
 but I agree that a CSS style language is a pretty reasonable approach
 to describe the layout
*/

:root {
  viewpoint: isometric;
}

.room {
  width: 12.5;
  height: 6.5;
  depth: 4;
  border: 0.3 solid midgray;
}

.star-wars-poster {
  parent: top-left;
  position: absolute;
  background: poster-image.png;
  top: 1;
  left: 0.5;
}

.window {
  parent: top-left;
  width: 50\%;
  height: 40\%;
  depth: 0.3;
  align: front-face;
  model: window-with-blind;
}

.bookcase {
  parent: top-right;
  left: 0;
  bottom: 0;
  model: bookcase;
  width: 2;
}

.corner-lighting {
  parent: top-right;
  width: 2;
  height: 1;
  align: top-face;
  lighting: conical-gradient(light blue 20\%, black);
  lighting-style: diffuse;
}
/* and so on */
```

After -

ID 60e1e9a6553e3500157a88ed

Before

It’s a bedroom.

The room is a rectangle with a longer width than height. The door is in the middle of the back wall.

The bed is a double bed placed in parallel to the door. Next to the bed are two small dressers with lamps on it and led lights behind it. The desk (shaped like a L) is on the far left of the room when entering the door. With the chair facing the door. In the corner right of the door is a closet next to it (between the door and closet) are two small posters. On the left of the door is a picture on the wall and a shoe compartment like closet, with some led behind it. There is a big window with roll-down curtains on the right wall next to the door. Next (left of the window) to that window is a star wars poster.

I hope this is sufficient enough.

After -

ID 60dddba6a3ae320015ef23da

Before

Consider a rectangular room. The room is 11 units wide and 7 units deep. Orient the room such that the north and south edges are 11 units in length and the east and west edges are 7 units in length. The room has red-brown tiles, each of which is 1 unit in width and depth. The walls are a dark gray.

Along the north wall, there is a door position such that its east edge is located at the halfway point of the north wall. If you know how wide a standard door is, then it may help to know for scale that the door is approximately 1.5 units in width (from this, you can presumably derive the size of the room, but if you don’t know how wide a standard door is, good luck).

In the north east corner, positioned against the north wall is a dresser, approximately 1.5 units wide and 0.75 units deep. On top of the dresser is a bookshelf of the same width as the dresser but only 0.5 units deep containing books and miscellaneous decorative items. On the top of the bookshelf are some sports balls in display cases. The dresser-bookshelf is mahogany in color.

Centered between the dresser-bookshelf and the door are two pieces of miscellaneous artwork, less than 1 unit in width and height, positioned on the wall so that one is above the other.

Centered along the west wall is a window 3 units wide, 2.5 units tall and 1.25 units off of the ground. Centered between the south edge of the west wall and the south edge of the window is a poster of Darth Vader, approximately 2 units tall and 1.5 units wide, positioned such that its top edge matches the top edge of the window.

Starting from the west edge, the south wall has a dresser, a queen bed, and a dresser positioned against it. The two dressers are identical; they are approximately 1 unit wide, 1 unit tall, and 0.5 units deep. Both have a small lamp on top and are colored a very dark brown (think furniture dark brown). The queen bed is positioned such that the headboard is against the south wall. It has gray sheets. It is a platform style frame with no box spring. There is no footboard.

In the south east corner is a “battle station”. It features an L-desk. The shape of the L desk can be described by forming an L with your *right* hand, looking at it such that you are viewing the back of your hand, and rotating your hand such that the two segments of the mirrored-L occupy the north and east edges. On the inside of the L is a gaming style rolling chair sitting on a hexagonal mat. On top of the north edge of the desk, from its west side, is a laptop, a desk speaker, and a keyboard & mouse on top of a desk pad. In the corner are three monitors arranged horizontally so that they form a slight curve centered on the sitting position of the desk. On the east edge, starting from the north is a set of headphones on a headphone holder, a matching desktop speaker, and then a computer tower.

Along the north wall, to the east of the door is a 3 unit wide, 1 unit tall piece of artwork, whose bottom edge is 2.5 units above the ground and whose west edge is 0.25 units away from the east edge of the door. Centered beneath the piece of art is a 1.25 unit wide, 0.5 unit deep, 1.5 unit tall shoe rack containing 5 shelves and 3 pairs of shoes.

After -

ID 60e47cb851e39400155b0dcc

Before

A room, rectangular, approximately 12 units long (x) by 7 units wide (y)

furniture locations:
bookshelf: (0,0) -> (1,0)
photos: (2,0)[wall]
door: (3,0)[wall]
painting: (5,0) -> (7,0) [wall]
shoe rack: (6,0)
window: (0,2) -> (0,4)[wall]
poster: (0,5) [wall]
bedside table: (0,6)
bed: (1,2) -> (4,6)
bedside table: (5,6)
desk: (3,8) -> (6,11)

After -

ID 60e627f3eb04ee001585226d

Before

A large and simplistic bedroom with a corner office, no open windows and illuminated by multiple lamps.

There's one large and two small posters and a wide painting or digital canvas on the wall.

After -

ID 60e88023179d2a0015e96f63

Before

Rectangular room with 4 walls,
North and South walls are the longest walls - roughly 11.5 tiles wide,
East and West walls are roughly 6.5 tiles wide,

Instructions go from Left to Right:
1) West Wall:
There is a bedside cabinet with a vertical rectangular lamp in the corner between West and South Walls about half a unit wide and slightly away from the South wall.
Darth Vader poster roughly 1 tile along at eye level,
Half a tile further along there is a projector screen slightly higher up than the poster,
The projector screen is roughly 3 tiles wide with similar height,
In the corner between West and North Wall there is a bookshelf with a chest of drawers at the bottom - this occupies roughly half a tile.

2) North Wall:
In the corner between the West and north walls there is a bookshelf, this bookshelf occupies roughly 2 tiles along the north wall.
There are two pictures hung above one another next to the bookshelf with a slight margin,
The two pictures are aligned in a column and each picture has a different width - they are center aligned.
There is another margin before a door, the door is 2 tiles wide and 4 tiles from the West wall,
The door does not go from floor to ceiling,
There is a sunset landscape photograph hanging from the North wall at eye level with roughly half a tile of margin from the door, this photograph is hung around eye level,
Below the landscape photograph there is a 5 tiered vertical shoe rack with an led booklight, it is placed at floor level and is 1 tile wide.
The shoe rack is center aligned with the landscape photograph,

3) East Wall:
In the corner between the East and South walls there is an L shaped computer desk with a depth of 2 tiles,
The computer desk occupies roughly 3.5 tiles along the East wall starting from the South wall corner.

4) South Wall:

The corner between the south wall and the west wall has a bedside cabinet (cabinet 1) with a rectangular lamp and led backlight,
this bedside cabinet is roughy 1 tile in width,
A double bed with backboard touching the South wall is a small margin east from cabinet 1,
The double bed is roughly 4 units wide and roughly 4.5 units long,
A small margin to the east from the beds east edge is cabinet 2, this cabinet is the same as cabinet 1 in terms of shape and has a rectangular lamp and led backlight,
roughly 2 tiles east of cabinet 2 is the start of a vertically squashed hexagonal mat with a width of roughly 3 units.
In the center of the mat is a computer desk.
In the corner between south and east wall is the computer desk.

In the center of the room there is a ceiling fan and light.

Further descriptors for the computer desk:

The computer desk runs roughly 3.5 units along the East wall from the corner between east and south walls,
The computer desk is an L shape with the corner of the L being at its most northern point along the east wall,
There is a computer tower, 3 monitors horizontally aligned on a single stand center aligned, a keyboard, mouse, headphone stand and headphones.

The west most point of the computer desk is where the laptop and laptop connector ports are station,
To the east of the laptop is the keyboard and mouse,
to the west of the L corner and at the most northern point of the computer desk is the monitor stand with the middle screen centered on the stand left most screen facing south, center screen South west and the right most screen south west west,
The most southern point of the computer desk is where the computer tower and computer connector port adapter is with and it faces east

After -

ID 610e3a94cc69bb001570a164

Before

There is a room. The room is about 3x6 meters large, and about 2.5m tall.

The room has tiling. The tiles are about 30x30 cm large. The tiles are of dark-ish brown texture. The tiles are connected via light-brown filling. The walls is dark beige.

There are doors on the longer side of the room, starting 2m from the left. The doors are white and about 2x1m in size.

There's a window on the wall that's to the left of the wall with the doors. The window is in the middle of the wall, white, and about 1.5 (W) x 1.2 (H) m in size. In the centre of the room there's a ceiling lamp with a ventilator. The lamp is shining in daylight tint (clear white).

The wall with the window has a poster to its left. The poster looks like a Fallout 3 poster, but in blue and purple colours.

The wall with the doors has cupboard at it's left end. The cupboard is about 2m high and 1m wide, wooden, with dark-ish glaze. The cubboard is a combination of a chest of drawers (1m high) and a bookcase. There's some stuff in the bookcase, and on top of it.

Above the cupboard, on the wall, in the corner where the wall meets the ceiling, there's a light-blue light mounted.

Between the cupboard and the doors, there are 2 small photos.

Next to the doors to the right, there's an ultra-wide TV mounted on the wall with a screensaver that you'd find on a MacBook.

Underneath the TV, there's a shoe-rack, about 0.5m wide and 1m high, black, metalic or plastic. The shoe rack has 4 levels, each level is see-through mesh. The bottom level has 2 shoe pairs, and the level above it has 1 pair of shoes. The shoerack has light-blue neon lighting mounted at its back (towards the wall) on the bottom and second-top levels.

In the corner that's furtherest away from the cupboard, there's a computer station - an L-shaped desk. The desk is oriented with one arm of the L-shape parallel with and closer to the wall with the door, and other arm is parallel with the wall with the window, but away from it. Both arms of the L-shaped desk are about 1.5m long. The desk is about 0.5m deep. The desk has thin black legs at each of the corners, about 1m high. The desk is black, with light-blue neon lighting along its rim. There's a laptop on the left, and a computer rig with 3 monitors in the corner. There's a gaming chair and a black hexagonal rug underneath the chair. Between the gaming setup and the wall with the window, there's a double-sized bed with a bed-side drawers on either side. The bed has white sheets, with light-grey and dark-grey covers on top of it. There's a soft dark-blue neon lighting installed underneath the bed. The bed's headrest also has the dark-blue lighting, and light-green one too at the top, but in the centre only.

Each bed-side drawers are about 0.3m wide, 0.6m tall and 0.2m deep, black, wooden, with light-orange backlight at its base. Both have square-ish bed-side lamps on top of them with clear with colour (same as the ceiling lamp) The drawers are about

After -

ID 60dcc028fbb87600156f9155

Before

A square bedroom rotated at 45 degrees. It contains an L shaped desk in the bottom corner with the opening to the bottom left wall. On the desk is a computer with four monitors and a laptop. There is a gaming chair in the opening of the desk. There is a bed in the left corner with night stands on either side of the headboard. There are orange lights coming from the bottom of the night stands. There is a hardwood floor. There is a bookshelf in the top corner with two upper shelves and a lower cabinet. On top of the bookshelf are some boxes. There is a window in the middle of the top left wall with a poster of Darth Vader to the left of it. There is a door in the middle of the top right wall with a poster to the right and two smaller posters to the left. There is a ceiling fan in the middle of the room.

After

There is a desk in the bottom corner with the computer at z 1 and also in the bottom corner. There is a chair in the opening of the desk. The desk should be rotated so that the opening is facing the left. There is a bed in the left corner facing the top right wall. There are night stands on either side of the headboard with their lights on. There is a bookshelf in the top corner. There is a ceiling fan in the middle of the room with its light on. There is a door in the middle of the top left wall with a shoe rack on the floor just to its right. The light on the shoe rack is on. The computer is powered on.

ID 60dcb2e0fbb87600156f90ad

Before

Aroud the perimeter, clockwise, from the door: Image of aurelia borealis above shoe-storage, empty wall, corner, emty wall, L-shaped desk with three curved monitors and tower, corner, gaming chair on mat in middle of L, Laptop and keyboard on desk. Nightstand with modern lamp, bed(kingsized), nightstand, corner, image, window with blinds, corner with books in storage, two small images, door again.
fan in middle of room, tiled brown floor

After

```
.bed{
 x:0
 y:1
}
.nightstand{
 y:0
 nr=1{x:0}
 nr=2{x:4}
}
.desk{
 x:8
 y:0
}
.fan{
 x:5
 y:5
 z:5
}
.door{
 x:3
 y:7
}
...
```

ID 60dccf3efbb87600156f9224

Before

An indoor room twice as long as it is tall and wide. The room is portrayed in an isometric view, with the floor and two walls (north and west) shown. The west wall, which is the shorter of the two, has a window which is horizontally centered, and vertically placed just slightly above center. As a percent of the whole wall, the window is about 50%. The window has a closed white shade. Centered between the left border of the window and what would be the south wall is a poster, which is about 75% as tall as the window, and is about 75% as wide as the space between the wall and the window. The north wall contains a bookcase, which is positioned in the corner between the north and west walls. It is about as tall as the highest point on the window. It extends out approximately two feet. The bookshelf is about three times as wide as it is deep. A space of wall between the right of the bookshelf and a door is about as long as the bookshelf. In this space, two small landscape pictures are mounted. A closed, white door is approximately in the middle of the north wall. To the right of the door, a large, horizontal painting is displayed. Below this painting is a black metal shoe rack with strip lighting on two of the five layers which it has to hold the shoes. There is empty space in the North-East corner of the room. The South-East corner of the room has an L-Shaped desk, on which three computer monitors are held by a single stand in the corner of the desk. The "L" shape is mirrored and placed in the south-east corner, so as to make the bottom of the letter touch a wall, and the side not. A swivel chair is at the desk, and it has a tower computer, laptop computer, keyboard and mouse, as well as a set of speakers on the desk. In the south-west corner of the room is a bed, with two bedside tables on either side of the headboard. The headboard is oriented to be against the south wall of the room. The back of the headboard is illuminated a blue and green color. The back of both bedside tables are illuminated a warm orange color. Both bedside tables have a lamp on top of them, both of which are illuminated a white color. The room has a centered ceiling fan and light. The light is illuminated a white color.

After

```
#room {
  size-x: 20;
  size-y: 10;
  size-z: 10;
}

#door {
  position-x: 4;
  position-z: 10;

  size-x: 1.5;
  size-z: 8;
  size-y: 0.01;
  color: #fff;

  rotation: -90deg;
}

#bookshelf {
  position-x: 0;
  position-z: 0;

  size-x: 2;
  size-z: 7;
  size-y: 1.5;

  rotation: -90deg;
}

#shoerack {
  position-x: 6;
  position-z: 0;

  size-x: 1.5;
  size-z: 4;
  size-y: 1;

  rotation: -90deg;
}

#shoerack > .led {
  power: on;
  color: lightblue;
}

#window {
  position-x: 0;
  position-z: 2;
  position-y: 4;

  size-x: 0.01;
  size-z: 5;
  size-y: 6;

  color: #fff;
}

#bed {
  position-x: 2;
  position-y: 10;

  rotation: 180deg;

/* Spent a bit over five minutes, stopping here */
```

ID 60dcdb5bfbb87600156f92ec

Before

A room in Isometric view, with the 2 closest walls missing. It has a window in the middle of the North-west wall, with a poster to the left of it. Against the South-west wall are in order starting from the north-west wall, but with their backs against the south west wall: A nightstand with a lamp on top with an orange glow at the bottom of the back, a bed with its headbord against the wall and another identical (including the lamps and glow) nightstand.

Against the South-east wall is a L-shaped desk going from the corner of the south east and south west walls first going along the south-east wall and then going towards the middle of the room. Inside the L is a desk chair. On the desk in order, starting at the side of the desk closest to the middle, A laptop, a speaker, a deskpad with mouse and keyboard. Headphones on a headphone stand, another speaker and a desktop computer. There are 3 screens around the corner of the desk, with the middle one right in the corner.

Against the North-East wall, starting at the North-West wall but with their backs against the North-East wall are in order: A shelve with books and balls in boxes on top, 2 small posters above eachother on the wall, a door, a larger poster of the northern lights and some shoes on some small shelves under that poster.

After

```
(x, y, z, [facing])

room(6.5, 11.5, 5)

nightstand1(0, 0, 0, NE)
lamp1(0, 0, 1, NE)
nightstand2(0, 6, 0, NE)
lamp2(0, 6, 1, NE)
bed(0, 1, 0, NE)
desk(3, 8, 0, SW)
chair(2, 9, 0, NE)
shelve(6.5, 0, 0, SW)
shoeshelve(6.5, 7, 0, SW)
```

ID 60dcc807fbb87600156f9207

Before

```
{
"size": "3m x 5m",
"long-sides": ["east", "west"],
"walls-nesw": [
  { "things-ltr": ["starwars-poster", "window-with-blinds"] },
  { "things-ltr": ["postcards", "door", "picture" },
  "none", "none" ],
"furniture": [
  { "object": "shelve", "description": "shelf with 3 rows, lamps on top, drawers at the bottom", "
     location": "north-east corner, facing west"},
  { "object": "shoe-rack", "description": "4 rows, illuminated", "location": "east wall .5m right
     to the door"},
  { "object": "desk", "description": "a corner desk equipped with a pc, 3 monitors, speakers,
     headphone-holder + headphones and a mac book", "location": "one side on south wall starting
     south-west corner, other side free in the room parallel to east and west wall" },
  { "object": "chair", "description": "office/gaming char with armrests. desk mat below", "
     location": "west wall next to the desk"},
  { "object": "bedside table", "description": "has a lamp on it", "location": "north-west corner
     "},
  { "object": "bed", "description": "140x200 bed with mattress and bed sheets", "location": "head
     to west wall; between bedside tables"},
  { "object": "bedside table", "description": "has a lamp on it", "location": "west wall next to
     bed"}
],
"other": ["ceiling lamp with fan in the middle of the room"]
}
```

After

The same as before.

ID 60dcb4a3fbb87600156f90b4

Before

It's a top-down, 45-degree isometric view of a room. The fore two walls and ceiling are hidden so we can see inside. The space is a bedroom with dim lighting.

The floor is made of square red/orange tiles (think saltillo, just with straight edges) that look to be about 1.5' x 1.5' in real-world size. The room is about 11.5 of these "long" (along the northwest-southeast axis) and 6.5 "wide". Since the half tiles are on the northeast and southeast edges of the space, let's treat the west corner as our origin for a coordinate system (x, y), where x is number of spaces traveling northeast from the origin and y southeast.

Furniture is placed as follows:

- A simple black nightstand with square IKEA lamp (lit) at (0, 0), facing northeast. 0.5 x 1 tile in size, about 1.5 tiles high
- An identical nightstand and lamp at (0, 5)
- A bed that spans the space between them, jutting out into the room such that the foot of the bed faces northeast. The bed is about 4.5 tiles long. It looks to be of a similar minimalist IKEA style, black, with simple headboard, white undersheets, a light gray topsheet, and mid-gray decorative sheet. Top of mattress about 1 tile high, headboard another 3/4 tile
- A wooden bookshelf/cabinet combo flush with the north corner, facing southwest. The base (cabinet) bit is 0.5 x 2 tiles in size and a little over 1 tile high, and the shelf on top tapers a bit towards the wall. Two inner shelves contain books. The "shelf" created by the top of the cabinet has various red knickknacks that are hard to make out. The very top of the shelf holds what looks to be square lamps (?)
- An L-shaped desk with simple black surface and cylindrical metal legs, somewhere between 1 and 1.5 tiles high. The L can be described by connecting the point (0, 10.5) to (2.5, 10.5) to (2.5, 7), then stroking that with a 1-tile-wide line (to the inside). On the desk are:
  - a triple monitor setup at the apex
  - a large keyboard and mouse mat (containing both) on the near left
  - a macbook pro on the far left
  - a black tower on the far right (no wires can be seen, though)
  - a headphone stand with over-ear headphones on the near right
  - a pair of small bookshelf speakers at mid-left and mid-right
– A gaming chair inside the L on top of a hexagonal pad. The pad is about 2 x 3 tiles in size and is hexagonal
- A multi-level shoe shelf at (6, 5.5) facing southwest. 5 shelves evenly divided over the height of about 2 tiles, 2 pairs of shoes on the bottom shelf, 1 pair on the shelf above that

A white 4-blade ceiling fan with lit lamp is placed in the center of the ceiling space.

There's a white door with no doorknob on the northeast wall at y coord 3, a tad less than 2 tiles wide. On the northwest wall is what appears to be a window — it's just a white panel, but the resolution of the picture might not be high enough to make out shades/drapes. The window spans (0, 2) to (0, 4), is about 2.5 tiles high, and is placed about 1.5 tiles off the ground.

Pictures are placed on both visible walls. A poster-sized picture is at the midpoint between the window and the southwest wall, aligned almost flush with the top of the window. There's a panoramic aurora borealis photo above the shoe shelf, and between the door and wooden shelf are two mid-sized photos stacked on top of each other. Can't make out the bottom one; top one might be the university of arizona 'A'.

There's orange underlighting behind both nightstands, subtle blue underlighting underneath the bed, and brighter blue-white underlighting behind the shoe shelf.

After

```
/* assumes we've been given pre-made objects to place, so I don't need to re-describe sizes,
     details, etc */

room {
  --size-x: 6.5;
  --size-y: 11.5;
}
end-table {
  --position-x: 0;
  --position-y: 0;
  --power: on;  /* let's assume underlighting is built into objects */
  --color: #fa6;
  --brightness: 60%;
}
#end-table-2 {
  --position-y: 5;
}
table-lamp {
  --position-z: 1.5;
  --power: on;
  --color: #fff;
  --brightness: 80%;
}
bed {
  --position-x: 0;
  --position-y: 1;
  --power: on;
  --color: #54a;
  --brightness: 20%;
}
L-desk {
  --position-x: 0;
  --position-y: 7;
  --rotation: 90deg;
}
chair {
  --position-x: 1;
  --position-y: 9;
  --rotation: -20deg;
}
shoe-rack {
  --position-x: 6;
  --position-y: 6;
  --power: on;
  --color: #eee4ff;
  --brightness: 70%;
}
shelf {
  --position-x 6;
  --rotation: 180deg;
}
door {
  --position-x: 6.5;
  --position-y: 3;
}
window {
  --position-x: 2;
  --position-z: 2;
}
fan {
  --position-x: 3;
  --position-y: 5;
  --position-z: 6;
  --power: on;
}
```

ID 60dcd49efbb87600156f924e

Before

- Room is rectangular, 7.5ft wide by 11.5 ft long (assuming floor tiles are 1ft by 1ft)
- Floor is tiled, with brown floor tiles (1ft by 1ft)
- For the rest of description, assume we are looking at room from the wide 11.5 ft wall.
- Large window 3.5ft wide and 3ft tall centered on left wall.
- Poster of character on left wall centered between window and close wall. 1.5ft tall by 2ft wide.
- Black bedside table on near left corner, facing towards far wall. 1.5ft wide by 1ft deep. Lamp on table.
- Bed immediately to the right of the bedside table. 3.5ft wide by 4.5 ft long (which makes me think my scale is off, but just going with it). Headboard against close wall, bed facing far wall. Gray sheets on bed
- Another identical bedside table immediately right of bed, facing far wall. Lamp on table.
- Brown dresser in far left corner of room. 2ft wide by 1ft deep. Two drawers, three shelves on top (shelves are less deep than drawers below). Boxes on top shelf, books on two shelves below, misc items on bottom shelf.
- White door to room on far wall 2 ft right of dresser.
- Two small pictures on wall between dresser and door. One inline with top of dresser, next below it and slightly larger.
- 5 shelf black shoe rack 1 ft right of door, two pairs of shoes on bottom shelf, 1 pair on shelf above.

- Painting of northern light scene on wall centered above shoe rack. 2.5ft wide by 1ft tall.
- Backward-L-shaped black computer desk in near-right corner of room. Bottom of backwards-L is against right wall. Blue LEDs around edge of desk.
- 3 side-by-side computer monitors on center corner of desk.
- Computer tower on desk, on bottom corner closet to wall.
- Headset on headset stand to the left of computer tower.
- Laptop on desk, near the top corner.
- Keyboard and mouse on desk mat to the right of laptop (under left-most computer monitor).
- Gaming chair inside the gap between backwards L and wall.
- Chair is on a black wide hexagonal mat with purple design on it.
- 4-blade white ceiling fan centered on ceiling of room.

After

```
iot-bed {
  --position-x: 2;
}

#desk-stand-2 {
  --position-x: 5;
}

iot-desk-stand iot-desk-light {
  --power: on;
}

iot-dresser {
  --rotation: 180deg;
  --position-y: 8;
}

iot-shoe-rack {
  --rotation: 180deg;
  --position-y: 8;
  --position-x: 8;
}

iot-desk {
  --rotation: 90deg;
  --position-x: 12;
}

iot-chair {
  --rotation: -45deg;
  --position-x: 12;
}

iot-ceiling-fan {
  --position-z: 8;
}
```

```
iot-ceiling-fan iot-light {
  --power: on;
}
```

ID 60dcc8fcfbb87600156f9213

Before

The room is rectangular with a door on a long side (for the sake of orientation, this is the east facing wall) and a window along the short side on the north facing wall, which has a white/grey roller blind pulled down fully.

The flooring used is a large (I estimate about 60x60cm) red/brown tile.

In the centre of the room, on the ceiling, is a fan with four blades and a light in the centre of the ceiling fan.

Also on the north wall, there is a painting or other graphical art to the left of the window.

In the corner of the north and east walls is a shelving unit, with its back against the east wall. The lower portion of the unit is deeper (extends into the room further) than the shelves above it. In this deeper section there appears to be drawers. There are two shelves above this deeper section, and the top of the unit (which could be described as a shelf).

The first surface of the unit - the top of the deeper section - has 5 items placed on it. Two are a red-orange colour, two are a dark purple, and the last is a beige ball.

The next shelf up is half filled with books (the right half) and there are two items placed on the remainder of the shelf.

The next shelf up is filled almost entirely with books, leaving space for about 2-3 books on the left side.

The very top of the unit has three items placed on it. These items take up the entire usable space of the top of the unit and are placed in a 1x3 line. Each item is close to, if not, square in footprint. The items on the outside edges look to be balls placed within glass boxes, while the item in the middle is an unidentifiable box shape.

To the right of the shelving unit, on the wall, but between the shelving unit and the door, there are two small pieces of artwork.

To the right of the door on the east wall, there is a large, landscape artwork of the northern lights. Underneath this artwork is a shoe rack. The shoe rack has 5 shelves, but there are only 3 pairs of shoes on it - two on the first shelf up and one on the second shelf up (on the left side). The shoe rack is only wide enough to accommodate two pairs of shoes per shelf. The shoe rack is illuminated by two strips of blue light on the back side (against the wall). These light strips line up with the bottom shelf and the fourth shelf up.

Along the south facing wall, and extending into the middle of the room, is a black, L shaped desk. The desk starts in the corner of the south and west walls, goes along the south wall about half way, then extends into the middle of the room about the same amount. Under the end of the desk in the middle of the room is a set of drawers. On the edges of the desk which touch walls, there is a strip of blue lighting. The items on the desk, starting from the side with the drawers, are as follows: a laptop (appears to be a macbook), open, has a pink wallpaper; a small cuboid speaker; a large deskpad; on the desk pad is a keyboard and a mouse; there are three monitors which are mounted on an arm which extends from the back corner of the desk, the wallpaper is the Cyberpunk 2077 art; a pair of over-ear headphones, resting on a stand; a second cuboid speaker; then finishing at the corner of the room is the PC case, sitting on the desk.

In the L of the desk, on the floor is a rounded-hexagonal shaped mat, on which a ‘gaming’ chair sits.

In the corner of the north and west walls, there is a bedside table with its back to the west wall. It has a lamp on top of it which has a long cuboid-shaped shade. There is an orange strip of light at the back, along the base of the table. There is a second one of these tables on the other side of the bed.

The bed is next along the west wall, with the headboard along the west wall. It is a double, or larger bed. The pillows and undersheet are white and there are two blankets/duvets over the bed. Both are grey in colour, one is darker than the other. The second, darker blanket/duvet is folded in half and is only over the end part of the bed. The bed’s headboard has a green strip of lighting behind it, against the wall. The strip of light is about half the width of the headboard, but is in the middle of the headboard at the top.

After

```
#bedside-table-1 {
    --position-x: 0;
    --position-y: 0;
}
#bed {
    --position-x: 0;
    --position-y: 1;
}
#bedside-table-2 {
    --position-x: 0;
    --position-y: 5;
}
#chair {
    --position-x: 1;
    --position-y: 8;
}
#desk {
    --position-x: 3;
    --position-y: 7;
}
#shelf {
    --position-x: 7;
    --position-y: 0;
    --rotation: 180deg;
}
#shoe-rack {
    --position-x: 7;
    --position-y: 6;
    --rotation: 180deg;
}
```

ID 60dcf5b2fbb87600156f95a6

Before

Walls are greyish brown, floor is brown wooden tile. Isometric view of the room, closest two walls are invisible. Room is rectangular, left wall is shorter than right wall. Left far wall has a window centered on it, with a portrait oriented poster centered between the window and the left edge of the wall at slightly higher than vertical center. Corner of left and right far wall contains a shelving unit facing directly away from the right wall, three items on the top, books filling the next shelf down from that, books filling the right hand side only of the next shelf down from that, various items on the next shelf down from that, followed by a set of closed drawers connecting the last shelf to the floor. The top of the shelf reaches around 3/4 of the way up the wall. Drawers make up around 1/3 of the furniture height. 40% along the far right wall starting on the left is the door. There are two landscape oriented posters centered between the shelves and the door, starting around half way up the wall vertically, with a short gap between them. There is a long landscape painting next to the door on the right with the same gap between it and the door as between the the posters and the door. Centered below the painting is a unit of 5 shelves with two shoes on the left on the second to bottom shelf and 4 shoes fully filling the bottom shelf. Along the back left wall starting at the corner between the front and back left walls, there is a rectangular black cube with a lamp centered on it, followed immediately by a bed with the headboard against the back left wall, then another identical cube with another lamp centered on it. The lamp has a rectangular base, a small gap, and then a rectangular lampshade, taking up around 1/3 of the width of the cube it is sitting on. The bed reaches across 70% of the room's width, the bed and cubes combined reach across around 50% of the room's length. The cubes with the lamps are only around the width of the bed's headboard and pillow (not visible). The bed has a white under sheet with grey over sheets. In the corner between the left and right front walls is a desk that runs along half the length of the front right wall, turns a 90 degree angle, and continues to reach around a quarter of the length of the room. There is nothing under the desk, the desk is very thin with thin black legs at every corner except the corners away from the walls, which reach the ground as a solid rectangular piece. There is a black rectangular computer tower closest to the room's corner, with a small black square next to it, followed by a bank of 3 monitors curving around the corner of the desk, and headphones on the desk near the inside corner of the desk on the same edge that the tower is on (the edge against the front right wall). On the edge sticking out into the middle of the room is a placemat, a keyboard on the placemat, a mouse to the right of the keyboard, anther black cube to the left of the edge of the monitor, and a small laptop to the left of everything else on the desk near the edge of the desk. In the alcove of the desk, facing directly away from the camera, is a gaming chair with no gaps in the back padding, with armpads reaching around half the length of the arm supports starting from the front and 6 pronged wheel array. Under the chair is a floor mat that does not lay flush with the back left wall, but has a tapered point sticking out from it that comes close to the wall in its middle. There is a 4 bladed ceiling fan centered on the ceiling.

After

side table at 0, 0; bed at 0, 1 through 0, 3 facing away from the wall; another side table at 0, 4, wooden shelves at 6, 0 and 6, 1 facing away from the wall; shoe stand at 6, 6; gaming desk from 0, 12 to 3, 12 and then it makes a right angle to 3, 8; gaming chair centered on a floor mat that covers 0, 8 to 2, 10 in a square; ceiling fan at the ceiling centered in the room

ID 60dd3dc984bb8c0015a4a1fa

Before

Room - 11.5 tiles long by 7.5 tiles wide. x = tiles along length, y = tiles along width

Nightstand 1x1 at x0,y0. Lamp on top. Poster on x0 wall above lamp
Bed 4x5 from x1,y0 to x4,y4. Headboard against wall
Nightstand 1x1 at x5,y0. Lamp on top
Window on wall along x0 line from y2 to y4
Bookshelf in corner 2x1 from x0,y6 to x1,y6
Photos on the wall immediately right of bookshelf
Door immediately right of photos (two tiles wide) at x3,y6.5 to x4,y6.5
Photo immediately right of door (three tiles wide)
Shoe rack centered below this photo, 1 tile wide
L shaped desk from x7,y3 to x10.5,y3 to x10.5,y0
Triple curved monitor on desk with tower, keyboard, laptop, headphone stand
Desk chair in the corner of the L desk

Floor is a reddish tile.
Walls are brown.
Ceiling light+fan centered in room

After

```
<room>
  <nightstand id="1"><lamp /></nightstand>
  <bed />
  <nightstand id="2"><lamp /></nightstand>
  <ceiling-light-with-fan />
  <window />
  <poster id="starwars" />
  <bookshelf />
  <door />
  <shoerack />
  <desk><computer /><chair /></desk>
</room>

room {
  --size-x: 11.5;
  --size-y: 7.5;
  --size-z: 7;
}

bed {
  --size-x: 3;
  --size-y: 5;
  --position-x: 1;
  --rotation: 180deg;
  color: gray;
}

nightstand#2 {
  --position-x: 5;
}
```

```
lamp {
  --position-z: 1;
  --power: on;
  --brightness: 100%;
}

ceiling-light-with-fan {
  --position-x: 5;
  --position-y: 3;
  --power: on;
  --fan-speed: 0%;
}

window {
  --size-y: 3;
  --size-z: 2;
  --position-y: 2;
  --position-z: 1;
  --rotation: -90deg;
}

poster#starwars {
  --position-z: 1;
  --rotation: -90deg;
}

bookshelf {
  --size-y: 2;
  --size-z: 3;
  --position-y: 6.5;
}

door {
  --size-y: 2;
  --size-z: 3;
  --position-y: 6.5;
  --position-x: 4;
}

shoerack {
  --position-x: 7;
  --position-y: 6.5;
}

desk {
  --size-x: 3;
  --size-y: 3;
  --position-x: 7;
}

computer {
  --position-x: 2;
  --position-y: 2
  --position-z: 1;
  --power: on;
}
```

```
chair {
  --rotation: -10deg;
}
```

ID 60dd5a8184bb8c0015a4a2e1

Before

Isometric top down view on single room.

Walls:
- West
-width: 0.5fr
-windows
-window
position: centered
width: 50%;
height: 50%
-accessories:
-poster
- North
-width: 1fr
-accessories:
-small image
-small image
-door
-wide image

The room contains:

- one cupboard placed in the west-north corner of the room
- one shoe-rack placed center of the north wall, slighly east of the door. its backlit blue
- one L-shaped desk, placed along the east wall, starting in the south east corner of the room with its long side reaching into the room. Desk contains from west to east: one macbook, small spaker, keyboard an mouse on a deskmat, 3 monitors, 1 headsetstand, second small speaker, one desktop PC case
- in front of desk: 1 gaming chair on a chair mat
- 1 double bed on the south wall, placed in the south-west corner of the room with the headpiece at the south wall. next to the headpiece are two symmetrically placed nightstands. the bed headframe is backlit green, the nightstands are backlit orange ant the bottom back side.
- mounted at the center of the rooms ceiling is a fan, light combo

After

```
iot-room {
  --width: 7;
  --length: 13;
  --height: 6;
}

iot-window {
   --position-x: 2;
   --position-y: 0;
   --position z: 2;
   --rotation: 90deg;
}

iot-poster{
   --width: 1;
   --height: 2;
   --position-x: 1;
   --position-y: 0;
   --position z: 2.5;
   --rotation: 90deg;
}

iot-cupboard {
   --position-x: 7;
   --position-y: 0;
   --rotation:180deg;
}

iot-poster:nth-of-type(2){
   --width: 0.5;
   --height: 0.25;
   --position-x: 7;
   --position-y: 3;
   --position z: 4;
   --rotation: 180deg;
}

iot-poster:nth-of-type(3){
   --width: 0.55;
   --height: 0.3;
   --position-x: 7;
   --position-y: 2.5;
   --position z: 4;
   --rotation: 180deg;
}

iot-door{
   --position-x: 7;
   --position-y: 4;
   --rotation: 180deg;
   --closed: true;
}

iot-poster:nth-of-type(4){
   --width: 2;
   --height: 1;
   --position-x: 7;
   --position-y: 7;
   --position z: 4;
   --rotation: 180deg;
}

iot-shoerack {
   --position-x: 7;
   --position-y: 8;
   --rotation: 180deg;
   --backlight: on;
   --backlight-color: blue;
}

iot-L-shape-desk{
   --position-x: 0;
   --position-y: 9;
   --backlight: on;
   --backlight-color: orange;
}

iot-bed{
   --position-x: 1;
   --position-y: 1;
   --backlight: on;
   --backlight-color: orange;
}
```

ID 60ddbba3a3ae320015ef234c

Before

a rectangular room (orthographic rendering, ceiling and two walls omitted)
- grey walls
– wall on the right is around 2x length of other wall, going from left to right:
— bookshelf, wooden
—- two parts:
–— lower part has drawers
–— upper part has books, three levels
—- 3 boxes with large balls in them on top
— two small paintings above each other at eye level
— white door, closed, plain
— painting of an aurora, wide (4:1)
— below painting a rack for shoes with 5 partially transparent levels, each fits two pairs of shoes
—- blue led illuminated, two strips
—- two pairs of shoes at the bottom, one above that
— unused space (1/4 of the wall)
– wall on the left (orthogonal to eachother since room is rectangular), going from left to right:
— darth vader poster at eye level
— window, medium large, white, can't see through (white cover?)
- brown square tiled floor, has following things going from top left to bottom right:
– bed area:
— wide bed with white sheets, light gray blanket, dark gray blanket (smaller)

— black headboard, blue and green illumination
— two nightstands, each:
—- orange/red bottom illumination
—- small lamp with tall square lampshade, on
– desk area:
— desk:
—- L-shaped, L-corner pointing towards right (room is rendered at 45 degree angle)
—- light blue led strip along edge
—- wide round feet on legs at each corner
—- items going from top to bottom:
—– macbook, open
—– small speaker
—– desk mat with keyboard and mouse (right side)
—– three screens, not curved, on monitor arm arranged in quarter-circle, cyberpunk 2077 wallpaper
—– gaming headset on stand
—– small speaker
—– computer, black
— gaming chair with headrest on rounded hexagon floor mat (black with purple design)
- ceiling has ceiling fan in center

After

```
<structure>
  <room>
    <wall>
      <bookshelf>
        <part>
          <drawer/>
          <drawer/>
        </part>
        <part>
          <shelf>
            <book/>
            <book/>
          </shelf>
          <shelf>
            <book/>
            <book/>
          </shelf>
        </part>
      </bookshelf>
      <painting/>
      <painting/>
      <door/>
      <part>
        <painting/>
        <rack>
          <level></level>
          <level>
            <light/>
          </level>
          <level></level>
          <level>
            <light/>
            <shoes/>
          </level>
          <level>
            <shoes/>
            <shoes/>
          </level>
        </rack>
      </part>
    </wall>
    <wall>
      <poster/>
      <window/>
    </wall>
    <floor>
      <part>
        <bed/>
        <head>
          <nightstand>
            <lamp/>
          </nightstand>
          <headboard/>
          <nightstand>
            <lamp/>
          </nightstand>
        </head>
      </part>
      <part>
        <part>
          <mat/>
          <chair/>
        </part>
        <desk>
          <macbook/>
          <speaker/>
          <mouse-mat>
            <keyboard/>
            <mouse/>
          </mouse-mat>
          <monitor-arm>
            <screen/>
            <screen/>
            <screen/>
          </monitor-arm>
          <headphone-stand>
            <headphones/>
          </headphone-stand>
          <speaker/>
          <computer/>
        </desk>
      </part>
    </floor>
    <ceiling>
      <fan>
        <lamp/>
      </fan>
    </ceiling>
  </room>
</structure>
```

```
<styling>
fan lamp {
  power: on;
}
rack light {
  power: on;
  color: blue;
}
rack level {
  position-z: calc(child-index())
}
door {
  color: white;
}
floor {
  material-color: brown;
  material-type: tiles;
}
part:has(bed) lamp {
  power: on;
}
part bed {
  position-x: 2;
}
... etc
</styling>
```

ID 60dddc72a3ae320015ef23db

Before

```
House (
 Size = 7 tiles, 12 tiles, 5 tiles
)
Floor (
 Color = Maroon
 Texture = Tiled
 Reflection = Glossy
)
Walls (
 Color = Gray
 Texture = Smooth
 Reflection = Glossy
)
Shelving1 (
 Color = Black
 Size = 0.5tl 1tl 2tl
 Shelves = 5
 Position = 1tl 7tl 0tl
 LightsColor = lightblue
 LightsPosition = first shelf, fourth shelf
 Lights = on
)
Shoes(
 PositionReference = Shelving
 Placement = 4 at the bottom shelf, 2 at the second to bottom shelf.
```

```
Painting (
 Position = 1tl 7tl 3tl
)
Desk (
 Shape = ( 1 1 1, 0 0 1, 0 0 1)
 Position = 6tl 9tl 0tl
 Size = 3tl 3tl 1tl
 Color = black
 Panels = top
 LightsColor = white
 LightsPosition = side
 Lights = On
)
Screen (
 PositionReference = Desktop
 Position = 1 3
 Screen = On
)
Keyboard (
PositionReference = Desktop
Position = 1 2
)
Laptop(
Position Reference = Desktop
Position = 1 1
Screen = on
Open = yes
Yaw = 135\degree{}
)
Chair(
Position = 6tl 9tl 0tl
Yaw = 315\degree{}
)
Mat(
Position = 6tl 9tl 0tl
)
Bed(
Position = 5tl 3tl 0tl
Size = 4tl 4tl 1tl
Color = white
)
Mattress1(
PositionReference = Bed
Color=LightGrey
)
Mattress2(
PositionReference=Bed
Color=DarkGray
(on top of Mattress1)
)
BedsideTableWithLamp1(
Position= 6tl 6tl 0tl
Color=black
LampColor=Black
LampLightColor=white
LampLight=On
)
BedsideTableWithLamp2(
Position= 6tl 1tl 0tl
Color=black
LampColor=Black
LampLightColor=white
LampLight=On
)
Poster(
Position= 6tl 1tl 3tl
)
Window(
Postion = 3tl 1tl 3tl
Color= white
Height=2tl
Width=3tl
)
Bookshelf(
Position = 1tl 2tl 0tl
Size= 1tl 2tl 3tl
Txture=wood
Color=brown
)
Door(
Color=white
Position=0tl 4tl 0tl
Height=4tl
Width=2tl
)
CeilingFanWithLamp(
Color=white
LightColor=white
Light=on
Position=3tl 6tl 5tl
)
```

After

```
Room{sizex:7; sizey:12; sizez:4; floorcolor:brown; wallcolor:gray;}
Desk{positionx:3; positiony:11; rotation=90deg} > display {position y:2 power:on rotation:120deg;
    rotation:}, > laptop{rotation:150deg}
Chair{positionx:2; positiony:10; rotation=270deg}
Mat{positionx:2; positiony:10)
Bed{positionx:3; positiony:3; rotation:180deg}
Lectern{power:on; lightcolor= white}[number="1"]{positiony:1}[number="2"]{positiony:6}
Bookshelf{positionx:7; positiony:7; rotation:90deg}
Window{positionx:4; positionz:2;}
Door{positionx:7;  rotation:90deg}
Poster{positionx:2;}
Painting{positionx:7; positiony:6; positionz:3; rotation:90deg}
Shelving{positionx:7; positiony:6; rotation:90deg}
CeilingFanWithLamp{positionx:4; positiony:5; positionz:1; power:on; lightcolor:white}
```

ID 60ddefdfa3ae320015ef2496

Before

A isometric view of a room about twice as long as it is wide. In the sort far wall is a window and on the same wall hangs a poster of a robot. In the other far wall is a door with a bookcase to the left for us. On the same wall is a painting with the northern lights. Against the long wall closest to us is a bed with two nightstands next to it. In the corner closest to us is a L shaped desk with a computer tower, 3 monitors, a keyboard and mouse and a laptop. In the space made by the L is a gaming chair.

After

```
room {
  size-xyz: 7,12,7;
}

book-case {
  position-xyz: 6,0,0;
  size-xyz: 1,2,5;
}

night-stand-1 {
  position-xyz: 0,0,0;
  size-xyz: 1,1,1;
}

night-stand-2 {
  position-xyz: 0,6,0;
  size-xyz: 1,1,1;
}

bed {
  position-xyz: 0,6,0;
  size-xyz: 5,4,1;
}

chair {
  position-xyz: 2,9,0;
  size-xyz: 1,1,3;
}

desk {
  position-xyz: 4,9,0;
  size-xyz: 4,3,2;
}
```

ID 60de377c2c08e9001539b3ab

Before

The floor has a dark amber colour and is divided in 11.5 times 6.5 square tiles.
The entrance doors spans (3|6.5) to (5|6.5).
A bed spans (1|0) to (5|0) with a padding of about 0.5.
There are shelves on tiles (0|0) and (0|5) with lamps on them.

A shelf with shoes is placed at the half-tile at (6|6).
A gaming chair is placed at (9|1) with a L-shaped table around it covering tiles
(9|2),(10|2),(11|2),(11|1),(11|0)

After

```
desk {

}
```

ID 60df1cc86138fb00154af407

Before

There is a rectangular room. On one of the longer walls, there is a door about 1/3 from a corner. Across from that door is a bed with feet facing the door (not very feng shui). From the inside of the room, facing the door, to the left of the door is a bookshelf. To the right of the door is a shoe rack and a panoramic painting of an aurora above it. To the right of the bed while inside the room and facing the door is a desk and chair. On the desk is a laptop, desktop, and three monitors all in landscape orientation, and side by side at the corner of the desk. The are small rectangular end tables with lamps at either end of the bed.

After

- There is a room (30(x), 15(y), 10(z))
- There is a bed at 5, 0, 0, facing the y direction
- There is a desk at 20, 0, 0, facing the y direction
- Upon that desk are monitors that round the corner
- At that desk is a chair rotated -45deg from the y direction
- There are end tables at 0,0,0 and 10,0,0, facing the y direction
- There is a bookshelf at 14,0,0
- There is a shoe rack at 14,18,0
- there is a door at 15,8,0
- there is a window at 0,3,9

ID 60e02cba84240200154c75f4

Before

There is a bed on the longer wall of the room. On both sides of the bed there are lamps on nightstands. To the right of the bed there is an L-shaped desk with a computer and a chair next to it. On the opposite wall, there is a door. To the right of the door there is a shoe rack. To the left there is a shelf with books on it. On the wall to the left of the bed there is a big window and a poster. There also is a fan on the ceiling.

After

Window on the left wall. On the near wall, against the left wall is a night stand with a lit lamp. Next to it is a bed, and a second night stand with a lamp on the other side of the bed. In the corner between the near wall and the right wall is a desk, with a chair next to it. On the far wall is a bookshelf next to the left wall, then doors and a shoe rack. In the middle of the room, on the ceiling, is a lit lighting fixture with a fan;

ID 60e09ae5ac638800156ed00f

Before

Begin room description:

A rectangular room, about 3 times as long as it is wide. The North and South walls are considered Short, east and West long. The ceiling is about 75% as high as the horizontal dimension of the short wall. The room has red stone tiles as floor, about 6.5 long on the short side and 11 or 12 on the long side? The walls are a dark beige.

Begin ingress/egress description:

A window is centered on the north wall with bland shades drawn. It takes up about half the width of the height and a third of the width of the wall.

A white door, about 80% the height of the room and 10% the width, sits on the East wall. Its right edge sits along the center line of the wall.

Begin interior of room:

A large bed, perhaps a full or queen size, is opposite the door against the West wall. its short side is along the wall. its right edge matches the right edge of the door. It does not reach the north wall.r It has a light grey comforter, wrinkled at the top as if someone got out of bed, up along 80% of it. There is another darker grey blanket folded along the bottom half of the bed, extending to about 90% of its length.

The bed has a simple thin headboard that may rise to about 40% the height of the room and is only a few inches thick

The bed and headboard are flanked by 2 small nightstands that come up to the height of the bed. If their widths were to be summed, it would be less than half the width of the bed. They are half as deep as they are wide. Atop each nightstand is a small brass lamp with a square base. An edge of their square base is about 50% the depth of the nightstand. After a few inches of a stem, they each have a square diffusing lampshade. The total height of the lampshade comes just past the height of the headboard.

A ceiling fan is centered on the room, with 4 white blades and a dome lamp. The ceiling fan is perhaps equal in width to the door. It has a white dome light. The entire fan/lamp hangs maybe 2 feet from the ceiling.

Begin computer desk description:

A L shaped computer desk is in the southwest corner. The edges of the L run along the south wall, and then extend north into the room. The south section of the L is on thin black legs, a few inches in diamter with round feet. The north end of the L has some sort of cabinet as its supposed. A laptop resembling a macbook sits atop this section of the desk, and the cabinet is slighter sider than the macbook. The desk is rimmed in a blue glow.

A nondescript black computer tower sits on the desktop, looks like an ATX mid tower. It is a foot from the west edge. 3 monitors are mounted in an arc, anchoring near the southeast corner. The mount sits on the east edge, but a foot or 2 from the south edge. The monitors are maybe a foot up from the desk surface

The arc of monitors is flanked by small squarish speakers. Speaker height is tallest dimension, followed by depth then width. They seem to have blue accents on the edges.

A large mousepad covers much of the desk surface, about twice as wide as the keyboard. It is a dark blue with a light blue trim.

A typical DX-Racer style gaming chair sits at the desk. It is on a mat in the shape of a hexagon, but with soft edges and it is wider than it is tall. It has a purple design on it.

End desk description

Begin interior along east wall:

A canvas painting or print of an aurora landscape is on the east wall, from about 60% of the height to 80%. It is maybe 4x as wide as it is tall. It is less than a foot to the south of the door.

Centered beneath it is a black wire shoe rack, that seems to be backlit blue. It is wide enough to accomodate 2 pairs of shoes on each of its 5 shelves. 2 pairs of sneakers sit on the bottom shelf, and 1 pair on the left side of the second from bottom.

Directly to the left of the door on the east wall are 2 small paintings, one about 55-65% of the height and another 70-75% of the height. It is unclear what is on them.

In the northeast corner is a piece of wooden furniture, extending to about 75% the height of the room. The bottom 1/3 seems to be drawers, the top portion is 3 shelves. The top 2 shelves have various books. On top are 3 balls running the length of it, each in what may be a glass commemorative case. The end ones may be white soccer or volleyballs, while the center one may be a football. The bottom shelf has various toys or other belongings.

Begin north wall:

To the left of the window is a poster with the profile of Darth Vader of the Star Wars franchise, stylized in front of an X Wing. It may be a promo poster for a video game. It takes up about half the horizontal space between the wall and window, and takes up maybe 50-80% of the height of the wall.

After

```
/* hello, I am sorry, this seems a bit out of my abilities/time available. I hope what I have
     contributed so far is useful to you */
```

ID 60e435126b67780015392b1f

Before

A room, 6.5 by 11.5 tiles big.
Wooden, tiled floor, grey walls.
Window in the middle of the left wall.
Door 3 tiles from the left, 2 tiles wide.
Ceiling fan with light bulb in the middle of the ceiling.
Bed 1 tile from the left wall, 3.5 tiles wide with a square, 1 tile wide night stand to either side, both topped with a lamp.
3.5 by 5 tiles gaming setup, with “L”-desk, chair, pc tower, 3 displays and a notebook.
1 by 0.5 tile wide shoe rack to the right of the door.

After

A room, 6.5 by 11.5 tiles big.
At 0x, 0y and 0z is a night stand, then a bed at 1y and then another night stand at 5y.
A night light is at the same xy as each the night stand, but with z1.
A “L”-desk at 0x, 7y and 0z with a chair at 0x, 8y and 0z.
A ceiling light at 3x, 5y and 10z.

ID 60ef390426b99e0015a5e6c9

Before

The room has grey smooth walls. The floor is lined with dark red tiles. On the left side of the room, there is a window and a poster on the wall. On the front side of the room there is a white door, colorful posters, a wooden bookcase and a mesh shoe rack. On the back side of the room there is a king size bed with gray sheets, two black bedside tables with lamps on them and a desk with three monitors and a notebook on them. There’s also a ceiling fan with a lightbulb.

After

```
iot-desk {
  --position-x: 10;
  --position-y: 5;
}

iot-bed {
  --position-x: 1;
  --position-y: 4;
}

iot-night-table {
  --position-y: 8;
}
```

ID 60f8389e4c03fd0015ad1ff9

Before

```
type DeviceGroup = {
  name: string;
  uuid: string;
};

type Light = {
  name: string;
  uuid: string;
  model: string;
  isOn: boolean;
  brightness: number;
  color?: string;
  group?: DeviceGroup;
};

type Scene = {
  name: string;
  uuid: string;
  devices: Array<Light>;
  deviceGroups?: Array<DeviceGroup>;
};
```

After

```
export type DeviceGroup = {
  name: string;
  uuid: string;
};

export interface Device {
  name: string;
  uuid: string;
  model: string;
  power: 'on' | 'off';
  deviceGroups?: Array<DeviceGroup>;
}

export interface Speaker extends Device {
  volume: number;
  isPlayingMedia: boolean;
}

export interface Light extends Device {
  brightness: number;
  color: string;
}

export interface Computer extends Device {
  isIdle: boolean;
}

export type Scene = {
  name: string;
  uuid: string;
  devices: Array<Device>;
  deviceGroups?: Array<DeviceGroup>;
};
```

```
export type Room = {
  name: string;
  uuid: string;
  temperature: number;
  isOccupied?: boolean;
  scenes?: Array<Scene>;
  devices?: Array<Device>;
  deviceGroups?: Array<DeviceGroup>;
};
```

ID 60fc2661259e7800150ee849

Before

The scene contains a rectangular room with sides approximately 15x30 ft in size. In the center of the ceiling is a fan with four blades. On the far, longer, wall, there is a shelf and dresser in the leftmost corner, followed by a pair of pictures. Next to the right, slightly to the left of the center of the room, is a door. Next is a lerger picture above a shoe rack.

On the near longer wall, a bed with two tables (one on each side) rests against the wall with the foot facing the door, with the leftmost dresser in the left corner. On the rightmost side (also against the corner), is an l-shaped desk. The l rests against the right shorter wall and extends out near the center of the wall, creating a small, semi-enclosed space where a gaming chair sits on top of a rug.

On the far (leftmost) shorter wall, there is a window and shade and a picture in the left/bottom corner.

After

```
shelf {
position: top left;
facing: bottom;
}

bed {
position: bottom left;
offset-left: width(bed-table);
facing: top;
}

bed-table {
position: bottom left;
  [number="2"] {
    offset: width(bed-table) + width(bed);
  }
}

fan {
position: ceiling center center;
light: on;
}

desk, chair {
position: bottom right;
}
```

```
chair {
offset-right: width(desk) / 2;
facing: 100deg;
}

shoe-rack {
positon: top center;
offset-left: .5m;
}

picture {
position-z: center;
wall: top;
}

picture[number=1] {
wall: left;
}
picture[number=2] {
offset-z: .2m;
}
```

References

[1] M. Schuhfuß, “Let there be light!,” Sep 2015. [Online]. Available: https://www.youtube.com/watch?v=ani_MOZt5_c

[2] J. Garza, D. J. Merrill, and S. Swanson, “Amalgam: Hardware hacking for web developers with style (sheets),” *Lecture Notes in Computer Science Web Engineering*, p. 315–330, 2019. [Online]. Available: https://cseweb.ucsd.edu/~jgarzagu/pdfs/Garza_ICWE2019_AmalgamHardwareHackingForWebDev.pdf

[3] T. Meier and U. Schemmert, “Applying web-technologies for device state processing in iot middleware,” in *Proceedings of the 19th International Middleware Conference (Posters)*, ser. Middleware ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 13–14. [Online]. Available: https://doi.org/10.1145/3284014.3284021

[4] S. Faulkner, A. Eicholz, T. Leithead, A. Danilo, and S. Moon, Eds., *HTML 5.2*. W3C, Jan 2021. [Online]. Available: https://www.w3.org/TR/html52/

[5] B. Bos, “CSS specifications,” 2021. [Online]. Available: https://www.w3.org/Style/CSS/specs.en.html

[6] J. Harband, S. yu Guo, M. Ficarra, and K. Gibbons, Eds., *ECMA-262, 12th edition*. ECMA International, June 2021. [Online]. Available: https://262.ecma-international.org/12.0/

[7] H. Wium Lie, “Cascading style sheets,” Ph.D. dissertation, University of Oslo, 2005. [Online]. Available: https://www.wiumlie.no/2006/phd/

[8] YesLogic Pty. Ltd., “Prince - convert HTML to PDF with CSS.” [Online]. Available: https://www.princexml.com/

[9] J. Robie, Ed., *What is the Document Object Model?* W3C, July 1998. [Online]. Available: https://www.w3.org/TR/WD-DOM/introduction.html

[10] S. Liu, “Number of software developers worldwide in 2018 to 2024,” September 2021. [Online]. Available: https://www.statista.com/statistics/627312/worldwide-developer-population

[11] Connectivity Standards Alliance, “Connected home over IP GitHub repository.” [Online]. Available: https://github.com/project-chip/connectedhomeip

[12] M. Lagally, R. Matsukura, T. Kawaguchi, K. Toumura, and K. Kajimoto, Eds., *Web of Things (WoT) Architecture 1.1.* W3C, November 2020. [Online]. Available: https://www.w3.org/TR/2020/WD-wot-architecture11-20201124/

[13] F. Heart, A. McKenzie, J. McQuillan, and D. Walden, Eds., *Completion Report, A History of the ARPANET, the First Decade.* DARPA, January 1978. [Online]. Available: https://www.w3.org/TR/2020/WD-wot-architecture11-20201124/

[14] Information Sciences Institute, University of Southern California, *RFC-791: Internet Protocol.* IETF, September 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc791

[15] V. G. Cerf and R. E. Kahn, “A protocol for packet network intercommunication,” *IEEE Trans on Comms*, vol. 22, no. 5, May 1974. [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall06/cos561/papers/cerf74.pdf

[16] R. Braden, Ed., *RFC-1122: Requirements for Internet Hosts – Communication Layers.* IETF, October 1989. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1122

[17] S. Deering and R. Hinden, *RFC-2460: Internet Protocol, Version 6 (IPv6) Specification.* IETF, December 1998. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2460

[18] S. Deering and R. Hinden, *RFC-8200: Internet Protocol, Version 6 (IPv6) Specification.* IETF, July 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8200

[19] D. Shin, “A socio-technical framework for internet-of-things design: A human-centered design for the internet of things,” *Telematics and Informatics*, vol. 31, no. 4, pp. 519–531, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736585314000185

[20] J. L. Borges, *The Garden of Forking Paths: Jorge Luis Borges.* Editorial Sur, 1941.

[21] V. Bush, “As we may think,” *The Atlantic*, July 1945. [Online]. Available: https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/

[22] Project Xanadu, “Project xanadu: Founded 1960, the original hypertext project,” May 2007. [Online]. Available: https://www.xanadu.net/

[23] L. Wedeles, “Prof. nelson talk,” *Vassar Miscellany News*, February 1965. [Online]. Available: https://web.archive.org/web/20031230152224/http://faculty.vassar.edu/mijoyce/MiscNews_Feb65.html

[24] T. Berners-Lee, “Information management: A proposal,” CERN, Tech. Rep., 1989.

[25] T. Berners-Lee, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its inventor.* Harper San Francisco, 1999.

[26] Google LLC, “Google chrome: The browser built by google.” [Online]. Available: https://www.google.com/intl/en_uk/chrome/

[27] Microsoft Corporation, “Microsoft edge: Choose the web browser that puts you first.” [Online]. Available: https://www.microsoft.com/en-us/edge

[28] Mozilla Corporation, “Firefox browser: Get the browser that protects what’s important.” [Online]. Available: https://www.mozilla.org/en-GB/firefox/new/

[29] Opera Norway, “Opera web browser: A browser for the real you.” [Online]. Available: https://www.opera.com/

[30] Wikipedia contributors, “History of wikipedia,” *Wikipedia*, September 2021. [Online]. Available: https://en.wikipedia.org/wiki/History_of_Wikipedia

[31] I. Casebourne, C. Davies, M. Fernandes, and N. Norman, *Assessing the accuracy and quality of Wikipedia entries compared to popular online encyclopaedias.* Epic, August 2012. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/2/29/EPIC_Oxford_report.pdf

[32] Web Hypertext Application Technology Working Group, “Custom elements,” in *HTML: Living Standard.* Web Hypertext Application Technology Working Group, September 2021. [Online]. Available: https://html.spec.whatwg.org/multipage/custom-elements.html

[33] Web Hypertext Application Technology Working Group, “Shadow tree,” in *DOM: Living Standard.* Web Hypertext Application Technology Working Group, August 2021. [Online]. Available: https://dom.spec.whatwg.org/#shadow-trees

[34] Web Hypertext Application Technology Working Group, “The template element,” in *HTML: Living Standard.* Web Hypertext Application Technology Working Group, September 2021. [Online]. Available: https://html.spec.whatwg.org/multipage/scripting.html#the-template-element

[35] Web Hypertext Application Technology Working Group, “Module-related host hooks,” in *HTML: Living Standard.* Web Hypertext Application Technology Working Group, September 2021. [Online]. Available: https://html.spec.whatwg.org/multipage/webappapis.html#integration-with-the-javascript-module-system

[36] Google LLC, “Angular.” [Online]. Available: https://angular.io/

[37] Facebook Inc., “React - a javascript library for building interfaces.” [Online]. Available: https://reactjs.org/

[38] E. You, “Vue.js: The progressive javascript framework.” [Online]. Available: https://vuejs.org/

[39] Web Hypertext Application Technology Working Group, “WHATWG - FAQ,” *whatwg.org*, 2021. [Online]. Available: https://whatwg.org/faq

[40] Web Hypertext Application Technology Working Group, “WHATWG GitHub repository.” [Online]. Available: https://github.com/whatwg

[41] International Organization for Standardization, *Information processing - Text and office systems - Standard Generalized Markup Language (SGML)*. International Organization for Standardization, October 1986.

[42] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Eds., *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C, November 2008. [Online]. Available: https://www.w3.org/TR/2008/REC-xml-20081126/

[43] M. Stachowiak, “Understanding HTML, XML and XHTML,” *WebKit Blog*, September 2006. [Online]. Available: https://webkit.org/blog/68/understanding-html-xml-and-xhtml/

[44] S. Pemberton, D. Austin, J. Axelsson, T. Çelik, D. Dominiak, H. Elenbaas, B. Epperson, M. Ishikawa, S. Matsui, S. McCarron, A. Navarro, S. Peruvemba, R. Relyea, S. Schnitzenbaumer, P. Stark, M. Altheim, F. Boumphrey, J. Burger, A. W. Donoho, S. Dooley, K. Hofrichter, P. Hoschka, M. Ishikawa, W. ten Kate, P. King, P. Klante, Z. Nies, D. Raggett, P. Schmitz, C. Wilson, T. Wugofski, and D. Zigmond, *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. W3C, January 2000. [Online]. Available: https://www.w3.org/TR/xhtml1/

[45] D. Austin, S. Peruvemba, S. McCarron, M. Ishikawa, M. Birbeck, M. Altheim, F. Boumphrey, S. Dooley, S. Schnitzenbaumer, and T. Wugofski, Eds., *XHTML Modularization 1.1 - Second Edition*. W3C, July 2010. [Online]. Available: https://www.w3.org/TR/xhtml-modularization/

[46] H. Wium Lie, *Cascading HTML style sheets – a proposal*. W3C, October 1994. [Online]. Available: https://www.w3.org/People/howcome/p/cascade.html

[47] E. J. Etemad and T. A. Jr., Eds., *CSS Cascading and Inheritance Level 3*. W3C, February 2021. [Online]. Available: https://www.w3.org/TR/css-cascade-3/

[48] S. Napoleon, *Practical FOSI for Arbortext Editor [Excerpts]*. FOSIexpert LLC, 2008. [Online]. Available: https://community.ptc.com/sejnu66972/attachments/sejnu66972/Arbortext/29167/1/practical-fosi-excerpts.pdf

[49] D. Kennedy, “DSSSL; an introduction,” *Microcomputer Systems, Inc.*, February 1997. [Online]. Available: https://web.archive.org/web/19990429162526/http://www.mcs.net/~dken/dslintro.htm

[50] R. Raisch, “Request for comments: Stylesheets,” *www-talk Mailing List*, June 1993. [Online]. Available: http://1997.webhistory.org/www.lists/www-talk.1993q2/0445.html

[51] P. Deutsch, *RFC-1952: GZIP file format specification version 4.3.* IETF, May 1996. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1952

[52] M. Andreessen, “Stylesheet language,” *www-talk Mailing List*, October 1993. [Online]. Available: http://1997.webhistory.org/www.lists/www-talk.1993q4/0266.html

[53] D. Raggett, A. L. Hors, and I. Jacobs, Eds., *HTML 4.01 Specification.* W3C, December 1999, ch. 4.1: Definitions. [Online]. Available: https://www.w3.org/TR/html40/conform.html#h-4.1

[54] S. Heaney, “Re: Stylesheet language,” *www-talk Mailing List*, October 1993. [Online]. Available: https://www.w3.org/Style/History/www.eit.com/www.lists/www-talk.1993q4/0295.html

[55] Z. Bloom, “The languages which almost became CSS,” *Eager Blog*, 2016. [Online]. Available: https://eager.io/blog/the-languages-which-almost-were-css/

[56] ISO/IEC JTC 1/SC 34: Document description and processing languages, *Information technology — Processing languages — Document Style Semantics and Specification Language (DSSSL).* International Office for Standardization, April 1996. [Online]. Available: https://www.iso.org/standard/18196.html

[57] R. B. Findler and J. Matthews, *Revised (6) Report on the Algorithmic Language Scheme*, M. Sperber, R. K. Dybvig, M. Flatt, A. V. Straaten, R. Kelsey, W. Clinger, and J. Rees, Eds. Cambridge University Press, September 2007. [Online]. Available: http://www.r6rs.org/final/r6rs.pdf

[58] XSLT Working Group, *The Extensible Stylesheet Language Family (XSL).* W3C, June 2017. [Online]. Available: https://www.w3.org/Style/XSL/

[59] B. Bos, “CSS & XSL,” *W3C: Web Style Sheets*, January 2021. [Online]. Available: https://www.w3.org/Style/CSS-vs-XSL.en.html

[60] B. Bos, “Why “variables” in CSS are harmful - an essay,” *W3C: Member pages*, September 2008. [Online]. Available: https://www.w3.org/People/Bos/CSS-variables

[61] M. Kay, Ed., *XSL Transformations (XSLT) Version 2.0 (Second Edition).* W3C, March 2021. [Online]. Available: https://www.w3.org/TR/2021/REC-xslt20-20210330/

[62] J. Robie, M. Dyck, and J. Spiegel, Eds., *XML Path Language (XPath) 3.1.* W3C, March 2017. [Online]. Available: https://www.w3.org/TR/2017/REC-xpath-31-20170321/

[63] A. Berglund, “Formatting objects,” in *Extensible Stylesheet Language (XSL) Version 1.1.* W3C, December 2006. [Online]. Available: https://www.w3.org/TR/xsl/#fo-section

[64] The Sass team, “Sass: Syntactically awesome style sheets.” [Online]. Available: https://sass-lang.com/

[65] T. Holowaychuk, “Stylus: Expressive, synamic, robust css.” [Online]. Available: https://stylus-lang.com/

[66] The core Less team, “Less: It’s CSS, with just a little more.” [Online]. Available: http://lesscss.org/

[67] 130 contributors at https://github.com/cssinjs/jss/graphs/contributors, “JSS.” [Online]. Available: https://cssinjs.org/?v=v10.4.0

[68] Marc G., J. Petroules, and M. Keller, Eds., *The AXR Project*. The AXR Project, October 2013. [Online]. Available: https://github.com/axr/specification

[69] M. Keller, “Manifesto: AXR project,” *AXR Blog*, December 2010. [Online]. Available: https://web.archive.org/web/20150810085821/http://axrproject.org/about/manifesto

[70] W3C Community Development Team, “CSS4 community group,” *W3C Community and Business Groups*, February 2020. [Online]. Available: https://www.w3.org/community/css4/

[71] A. Deveria and L. Schoors, “Can i use... support tables for HTML5, CSS3, etc.” [Online]. Available: https://caniuse.com

[72] Mozilla and individual contributors, “CSS: Cascading style sheets,” *MDN Web Docs*, 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS

[73] D. Flanagan, *JavaScript, the Definitive Guide, 6th Edition*. O’Reilly Media, Inc., 2011.

[74] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 16 Edition*. Oracle, February 2021. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se16/jls16.pdf

[75] Adobe Systems Incorporated, *SWF File Format Specification, Version 10*. Adobe Systems Incorporated, 2008. [Online]. Available: https://www.adobe.com/content/dam/acom/en/devnet/pdf/swf-file-format-spec-v10.pdf

[76] Wikipedia contributors, “Adobe shockwave,” *Wikipedia*, June 2021. [Online]. Available: https://en.wikipedia.org/wiki/Adobe_Shockwave

[77] A. Rossberg, Ed., *WebAssembly Specification, Release 1.1 (Draft 2021-09-16)*. WebAssembly Community Group, September 2021. [Online]. Available: https://webassembly.github.io/spec/core/_download/WebAssembly.pdf

[78] Google LLC, *Dart Programming Language Specification, 5th edition*. Google LLC, April 2021. [Online]. Available: https://dart.dev/guides/language/specifications/DartLangSpec-v2.10.pdf

[79] The Rust Team, *The Rust Reference.* The Rust Team, September 2021. [Online]. Available: https://doc.rust-lang.org/reference/

[80] OpenJS Foundation, "Node.js." [Online]. Available: https://nodejs.org/

[81] G. Williams, "Espruino - javascript for microcontrollers." [Online]. Available: http://www.espruino.com/

[82] Q. Stafford-Fraser. [Online]. Available: https://en.wikipedia.org/wiki/File:Trojan_Room_coffee_pot_xcoffee.png

[83] "World Wide Web Consortium (W3C)." [Online]. Available: https://www.w3.org/

[84] Z. Kis, D. Peintner, C. Aguzzi, J. Hund, and K. Nimura, Eds., *Web of Things (WoT) Scripting API.* W3C, November 2020. [Online]. Available: https://www.w3.org/TR/wot-scripting-api/

[85] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "WoT scripting API," in *Web of Things (WoT) Architecture.* W3C, April 2020. [Online]. Available: https://www.w3.org/TR/wot-architecture/#sec-scripting-api

[86] Krellian Ltd., "WebThings." [Online]. Available: https://webthings.io/

[87] S. Kaebisch, T. Kamiya, M. McCool, V. Charpenay, and M. Kovatsch, Eds., *Web of Things (WoT) Thing Description.* W3C, April 2020. [Online]. Available: https://www.w3.org/TR/wot-thing-description/

[88] M. W. Murhammer, O. Atakan, S. Bretz, L. R. Pugh, K. Suzuki, and D. H. Wood, "The open systems interconnect (OSI) model," in *TCP/IP Tutorial and Technical Overview,* October 1998, pp. 9–11.

[89] IEEE, *Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)).* IEEE, August 2017. [Online]. Available: https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf

[90] M. W. Murhammer, O. Atakan, S. Bretz, L. R. Pugh, K. Suzuki, and D. H. Wood, "Address resolution protocol (ARP)," in *TCP/IP Tutorial and Technical Overview,* October 1998, pp. 68–72.

[91] T. Berners-Lee, R. Fielding, and L. Masinter, *RFC-3986: Uniform Resource Identifier (URI): Generic Syntax.* IETF, January 2005. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3986

[92] T. Nelson, "Ted nelson's computer paradigm, expressed as one-liners," *Xanadu Australia: Archived Keio Home Page of Ted Nelson,* January 1999, Available: https://xanadu.com.au/ted/TN/WRITINGS/TCOMPARADIGM/tedCompOneLiners.html.

[93] L. Sauermann, "Using the semantic web [presentation]," *W3C Semantic Web*, April 2008. [Online]. Available: https://www.posccaesar.org/svn/pub/SemanticDays/2008/LeoSauermann-semantic_web_in_use_semwebdaysnorway2008.pdf

[94] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary Perspectives on Complex Systems*, 2017, pp. 85–113.

[95] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P.-A. Champin, and N. Lindström, *JSON-LD 1.1*, G. Kellogg, P.-A. Champin, D. Longley, M. Sporny, and M. Lanthaler, Eds. W3C, July 2020. [Online]. Available: https://www.w3.org/TR/json-ld11/

[96] W3C, "Linked data," *W3C Standards: Semantic Web*, 2015. [Online]. Available: https://www.w3.org/standards/semanticweb/data

[97] S. Harris, A. Seaborne, and E. Prud'hommeaux, Eds., *SPARQL 1.1 Query Language*. W3C, March 2013. [Online]. Available: https://www.w3.org/TR/sparql11-query/

[98] R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J. J. Carroll, and B. McBride, Eds., *RDF 1.1 Concepts and Abstract Syntax*. W3C, February 2014. [Online]. Available: https://www.w3.org/TR/rdf11-concepts/

[99] R. L. Ackoff, "From data to wisdom," *Journal of applied systems analysis*, vol. 16, no. 1, pp. 3–9, 1989.

[100] J. Uri, "50 years ago: Apollo 13 crew returns safely to earth," *NASA Online*, April 2020. [Online]. Available: https://www.nasa.gov/feature/50-years-ago-apollo-13-crew-returns-safely-to-earth

[101] D. Gelernter, *Mirror Worlds, or: The Day Software Puts the Universe in a Shoebox ... How it Will Happen and What it Will Mean*. Oxford University Press, Inc., 1991.

[102] M. Grieves, *Virtually Perfect: Driving Innovative and Lean Products through Product Lifecycle Management*. Space Coast Press, 11 2011.

[103] S. Locks, "Know the difference between IoT and telecontrol," *Sofía*, March 2018. [Online]. Available: https://medium.com/sofialocks-en/know-the-difference-between-iot-and-telecontrol-d9489f8ba175

[104] Wikipedia contributors, "SCADA," *Wikipedia*, September 2021. [Online]. Available: https://en.wikipedia.org/wiki/SCADA

[105] I. Fette and A. Melnikov, *RFC-6455: The WebSocket Protocol*. IETF, December 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455

[106] Wikipedia contributors, "Polling (computer science)," *Wikipedia*, November 2020. [Online]. Available: https://en.wikipedia.org/wiki/Polling_(computer_science)

[107] J. Klensin, *RFC-5321: Simple Mail Transfer Protocol.* IETF, October 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5321

[108] J. Oikarinen and D. Reed, *RFC-1459: Internet Relay Chat Protocol.* IETF, May 1993. [Online]. Available: https://tools.ietf.org/html/rfc1459

[109] C. Cooper, “PointCast’s painful post-mortem,” *ZDNet*, 1999. [Online]. Available: https://www.zdnet.com/article/pointcasts-painful-post-mortem/

[110] M. Belshe and R. Peon, *RFC-7540: Hypertext Transfer Protocol Version 2 (HTTP/2)*, M. Thomson, Ed. IETF, May 2017. [Online]. Available: https://tools.ietf.org/html/rfc7540

[111] K. P. Birman and T. A. Joseph, Eds., *Exploiting Virtual Synchrony in Distributed Systems.* DARPA, February 1987. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA221858.pdf

[112] W3C qa-dev group, *RSS 2.0 Specification.* W3C, 2002. [Online]. Available: https://validator.w3.org/feed/docs/rss2.html

[113] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, Eds., *MQTT Version 5.0.* OASIS, March 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[114] J. Teicher, “The little-known story of the first IoT device,” *IBM Blogs*, February 2017. [Online]. Available: https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/

[115] M. Weiser, “The computer for the 21st century,” *Scientific American*, vol. 265, no. 3, pp. 94–105, 1991. [Online]. Available: http://www.jstor.org/stable/24938718

[116] M. Weiser and J. S. Brown, “Designing calm technology,” *PowerGrid Journal*, vol. 1, no. 1, pp. 75–85, 1996.

[117] M. Weiser and J. S. Brown, “The coming age of calm technology,” in *Beyond calculation.* Springer, 1997, pp. 75–85.

[118] M. P. Aylett and A. J. Quigley, “The broken dream of pervasive sentient ambient calm invisible ubiquitous computing,” in *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 425–435. [Online]. Available: https://doi.org/10.1145/2702613.2732508

[119] K. Ashton, “That ‘internet of things’ thing,” *RFID Journal*, June 2009. [Online]. Available: http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf

[120] J. Elder, “How kevin ashton named the internet of things,” *Avast Blog*, August 2019. [Online]. Available: https://blog.avast.com/kevin-ashton-named-the-internet-of-things

[121] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic, “People, places, things: Web presence for the real world,” in *Proceedings Third IEEE Workshop on Mobile Computing Systems and Applications*, 2000, pp. 19–28.

[122] qr-code-generator.com, [Edited from original]. [Online]. Available: https://www.qr-code-generator.com/wp-content/themes/qr/new_structure/assets/media/images/qr_codes_on/gallery/billboards/v2/billboard-05.png

[123] T. Boyle, January 2013. [Online]. Available: https://qfuse.com/blog/wp-content/uploads/2013/01/MatressQR_2-800x600.jpg

[124] “Yokogawa electric corporation.” [Online]. Available: https://www.yokogawa.com/

[125] Honeywell, “Industrial automation and control solutions from honeywell.” [Online]. Available: https://www.honeywellprocess.com/

[126] M. Larabel, “The linux kernel enters 2020 at 27.8 million lines in git but with less developers for 2019,” *Phoronix*, January 2020. [Online]. Available: https://www.phoronix.com/scan.php?page=news_item&px=Linux-Git-Stats-EOY2019

[127] Apple Inc., “Home.” [Online]. Available: https://www.apple.com/uk/ios/home/

[128] DosLab Electronics, LLC, “Demux.” [Online]. Available: https://doslabelectronics.com/Demux.html

[129] dosdude1, “Permanently disable 2011 15"/17" macbook pro dedicated GPU - gMux IC bypass,” *MacRumours*, April 2019. [Online]. Available: https://forums.macrumors.com/threads/permanently-disable-2011-15-17-macbook-pro-dedicated-gpu-gmux-ic-bypass.2134019/post-27296142

[130] The openHAB Community and the openHAB Foundation e.V., “Who we are: Our vision and philosophy,” *openHAB Website*, 2021. [Online]. Available: https://www.openhab.org/about/who-we-are.html

[131] Home Assistant, Inc., “Home assistant.” [Online]. Available: https://www.home-assistant.io/

[132] O. Ben-Kiki, C. Evans, and I. döt Net, Eds., *YAML Ain’t Markup Language (YAML) Version 1.1.* YAML Language Development Team, November 2005. [Online]. Available: https://yaml.org/spec/1.1/current.html

[133] IFTTT, "IFTTT: Do more with the things you love." [Online]. Available: https://ifttt.com/

[134] OpenJS Foundation and Node-RED contributors, "Node-red." [Online]. Available: https://nodered.org/

[135] "OpenJS Foundation." [Online]. Available: https://openjsf.org/

[136] P. Wieland. [Online]. Available: https://flows.nodered.org/node/node-red-contrib-saprfc

[137] E. Stark, F. Schindler, E. Kučera, O. Haffner, and A. Kozáková, "Adapter implementation into mozilla webthings IoT platform using javascript," in *2020 Cybernetics & Informatics (K & I)*, 2020, pp. 1–7.

[138] Google LLC, "Google nest, build your connected home." [Online]. Available: https://store.google.com/gb/category/connected_home?hl=en-GB

[139] M. Wohlsen, "What google really gets out of buying nest for $3.2 billion," *Wired Magazine (Online)*, January 2014. [Online]. Available: https://www.wired.com/2014/01/googles-3-billion-nest-buy-finally-make-internet-things-real-us/

[140] Thread Group, "Thread benefits," *Thread*, 2022. [Online]. Available: https://www.threadgroup.org/What-is-Thread/Thread-Benefits

[141] Signify Holding, "Smart lighting: Philips hue." [Online]. Available: https://www.philips-hue.com/en-gb

[142] J. Khan, "Philips launching app controlled 'hue personal wireless lighting' bulbs at apple stores tomorrow," *9TO5Mac*, October 2012, Available: https://web.archive.org/web/20121031233442/https://9to5mac.com/2012/10/29/philips-launching-app-controlled-hue-personal-wireless-lighting-bulbs-at-apple-stores-tomorrow/.

[143] Google LLC, "Google home - apps on google play." [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.chromecast.app

[144] Google LLC, "Integrate with google nest," *Device Access*, 2022. [Online]. Available: https://developers.google.com/nest/device-access

[145] Amazon.com, Inc., "Amazon.co.uk: Meet alexa." [Online]. Available: https://www.amazon.co.uk/gp/browse.html?node=12728352031

[146] Amazon.com, Inc., "Amazon.co.uk: Smart speakers." [Online]. Available: https://www.amazon.co.uk/gp/browse.html?node=21832611031

[147] Amazon Web Services, Inc., "What is AWS?" [Online]. Available: https://aws.amazon.com/what-is-aws/

[148] Google LLC, “Cloud computing services: Accelerate your transformation with google cloud.” [Online]. Available: https://cloud.google.com/

[149] Apple Inc., “Homekit: Developing apps and accessories for the home.” [Online]. Available: https://developer.apple.com/homekit/

[150] Apple Inc., “Siri: Siri does more than ever. even before you ask.” [Online]. Available: https://www.apple.com/uk/siri/

[151] IFTTT Inc., “What is IFTTT?” November 2010. [Online]. Available: https://web.archive.org/web/20101115034046/http://ifttt.com/wtf

[152] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, *RFC-4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks.* IETF, September 2007. [Online]. Available: https://tools.ietf.org/html/rfc4944

[153] IEEE P802.15 Working Group, *IEEE 802.15.4-2020 - IEEE Standard for Low-Rate Wireless Networks.* The Institute of Electrical and Electronics Engineers, Inc., July 2020. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9144691

[154] E. Kim, D. Kaspar, N. Chevrollier, and J. Vasseur, *Design and Application Spaces for 6LoWPANs.* IETF, July 2011. [Online]. Available: https://tools.ietf.org/html/rfc7540

[155] A. Fabre, K. Martinez, G. Bragg, P. Basford, J. Hart, S. Bader, and O. Bragg, “Deploying a 6LoWPAN, CoAP, low power, wireless sensor network,” in *ACM Conference on Embedded Networked Sensor Systems*, November 2016.

[156] Connectivity Standards Alliance, “Matter.” [Online]. Available: https://buildwithmatter.com

[157] Connectivity Standards Alliance, “Connectivity standards alliance: Building the foundation and future of the IoT.” [Online]. Available: https://csa-iot.org/

[158] Snap One, LLC, “Home automation and smart home control: Control4.” [Online]. Available: https://www.control4.com/

[159] Crestron Electronics Inc., “Crestron home.” [Online]. Available: https://www.crestron.com/Products/Market-Solutions/Crestron-Home

[160] Savant Systems, Inc., “Smart home automation: Savant.” [Online]. Available: https://www.savant.com/

[161] D. Raggett, “Launching the web of things interest group,” *W3C WoT Interest Group Blog*, January 2015. [Online]. Available: https://www.w3.org/blog/2015/01/launching-the-web-of-things-interest-group/

[162] D. Guinard and V. Trifa, “Web of things: Architecting the web of things, for techies and thinkers!,” 2007. [Online]. Available: https://webofthings.org/

[163] V. Trifa, D. Guinard, and D. Carrera, Eds., *Web Thing Model*. W3C, August 2015. [Online]. Available: https://www.w3.org/Submission/wot-model/

[164] Thingweb, “thingweb.” [Online]. Available: http://www.thingweb.io/

[165] W3C WoT Working Group, “Web of things working group - participants,” *W3C WoT Working Group*, 2021. [Online]. Available: https://www.w3.org/groups/wg/wot/participants

[166] W3C WoT Working Group and Interest Group, March 2021. [Online]. Available: https://www.youtube.com/watch?v=WMFXg-kni0U

[167] J. T. Collins and E. Korkan, “Relationship to project connected home over IP,” *W3C WoT Working Group GitHub: WoT Architecture*, September 2020. [Online]. Available: https://github.com/w3c/wot-architecture/issues/536

[168] E. Bunker and J. Harros, “Zigbee alliance: Getting started with project connected home over IP (webinar),” December 2019. [Online]. Available: https://www.smarthomepoint.com/wp-content/uploads/2020/01/Getting-Started-with-Project-CHIP-final.pdf

[169] I. Meyer, V. Zaluski, and K. Mackintosh, “Metaphorical internet terms: A conceptual and structural analysis",” *Terminology. International Journal of Theoretical and Applied Issues in Specialized Communication*, vol. 4, no. 1, pp. 1–33, 1997.

[170] T. Colburn and G. Shute, “Metaphor in computer science,” *Journal of Applied Logic*, vol. 6, no. 4, pp. 526–533, 2008, the Philosophy of Computer Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570868308000463

[171] B. Doub, K. Hayden, , and S. C. Lott, “Guide to the community memory records,” *Online Archive of California*, September 2015. [Online]. Available: http://pdf.oac.cdlib.org/pdf/camvchm/102733953-Community-Memory.pdf

[172] Apple Computer, Inc. [Online]. Available: https://upload.wikimedia.org/wikipedia/en/5/50/Apple_Macintosh_Desktop.png

[173] Apple, Inc. [Online]. Available: https://apiumhub.com/tech-blog-barcelona/skeuomorphic-design/

[174] Nullsoft, Inc. [Online]. Available: https://archive.org/details/winampskin_kenwood

[175] J. Clover, May 2013. [Online]. Available: https://www.macrumors.com/2013/05/01/apple-engineers-working-overtime-on-ios-7s-deforstallization/

[176] N. Shadbolt, K. O’Hara, D. De Roure, and D. Hall, *The Theory and Practice of Social Machines*. Springer Cham, 2019.

[177] K. Beck, M. Beedle, M. A. van Bennekum, M. A. Cockburn, M. W. Cunningham, M. M. Fowler, M. J. Grenning, M. J. Highsmith, M. A. Hunt, M. R. Jeffries, M. J. Kern, M. B. Marick, M. R. C. Martin, M. S. Mellor, M. K. Schwaber, M. J. Sutherland, and M. D. Thomas, "Principles behind the agile manifesto," *Manifesto for Agile Software Development*, 2001. [Online]. Available: http://agilemanifesto.org/principles.html

[178] A. Madaan, J. Nurse, D. De Roure, K. O'Hara, W. Hall, and S. Creese, "A storm in an IoT cup: The emergence of cyber-physical social machines," University of Southampton, Tech. Rep., 09 2018.

[179] S. Jobs, "Top 100 - a," *Internal Memo*, October 2010. [Online]. Available: https://twitter.com/TechEmails/status/1428400060019068933/photo/1

[180] reddit inc., "Buy it for life: Durable, quality, practical." [Online]. Available: https://www.reddit.com/r/BuyItForLife/

[181] Wikipedia contributors, "Survivorship bias," *Wikipedia*, October 2021. [Online]. Available: https://en.wikipedia.org/wiki/Survivorship_bias

[182] BERG Ltd., "Hello, little printer," *Berg Cloud*, December 2011. [Online]. Available: https://web.archive.org/web/20120915003521/http://bergcloud.com/

[183] Nord Projects Ltd., "Little printers, a friendly new messaging app and cloud platform." *Nord Projects*, 2019. [Online]. Available: https://nordprojects.co/projects/littleprinters/

[184] M. Webb and teh, "GitHub: Sirius server." [Online]. Available: https://github.com/genmon/sirius

[185] C. Cimpanu, "AWS outage impacts thousands of online services," *ZDNet: Tech Industry*, November 2020. [Online]. Available: https://www.zdnet.com/article/aws-outage-impacts-thousands-of-online-services/

[186] D. Zeng, S. Guo, and Z. Cheng, "The web of things: A survey (invited paper)," *JCM*, vol. 6, pp. 424–438, 09 2011.

[187] D. Winseck, "Netscapes of power: Convergence, network design, walled gardens, and other strategies of control in the information age," in *Surveillance as Social Sorting: Privacy, risk and digital discrimination*, D. Lyon, Ed., 2003, pp. 176–198.

[188] D. Weck, D. Raggett, D. Glazman, and C. Santambrogio, Eds., *CSS Speech Module*. W3C, March 2020. [Online]. Available: https://www.w3.org/TR/css-speech-1/

[189] Control Protocols Working Group, *ANSI E1.11 – 2008 (R2018): Entertainment Technology - USITT DMX512-A: Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories*. Entertainment Services and Technology Association (ESTA), April 2018. [Online]. Available: https://tsp.esta.org/tsp/documents/docs/ANSI-ESTA_E1-11_2008R2018.pdf

[190] M. Marschalko, “Using XML and CSS for electronic projects,” *Mate Marschalko*, 2017.

[191] A. Owen, “Skills matter skillscast: CSS and the internet of things,” October 2015. [Online]. Available: https://skillsmatter.com/skillscasts/6750-css-and-the-internet-of-things

[192] D. Guinard, V. Trifa, and E. Wilde, “A resource oriented architecture for the web of things,” in *2010 Internet of Things (IOT)*, 2010, pp. 1–8.

[193] Wikipedia contributors, “Representational state transfer,” *Wikipedia*, May 2022. [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer

[194] Wikipedia contributors, “Remote procedure call,” *Wikipedia*, May 2022. [Online]. Available: https://en.wikipedia.org/wiki/Remote_procedure_call

[195] D. Guinard, I. Ion, and S. Mayer, “In search of an internet of things service architecture: Rest or ws-*? a developers’ perspective,” in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 104, 05 2012.

[196] “React native: Learn once, write anywhere.” [Online]. Available: https://reactnative.dev/

[197] T. Wang, L. Wu, and Z. Lin, “The revival of mozilla in the browser war against internet explorer,” in *Proceedings of the 7th International Conference on Electronic Commerce*, January 2005, pp. 159–166.

[198] ISO/IEC JTC 1 Information technology, *ISO/IEC 7498-1:1994: Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. International Office for Standardization, November 1994. [Online]. Available: https://www.iso.org/standard/20269.html

[199] Telecommunication Standardization Sector of ITU, *X.200: Information technology - Open Systems Interconnection - Basic Reference Model: The basic model*. International Telecommunication Union, July 1994. [Online]. Available: https://www.itu.int/rec/T-REC-X.200-199407-I/en

[200] W. siong Tan, D. Liu, and R. Bishu, “Web evaluation: Heuristic evaluation vs. user testing,” *International Journal of Industrial Ergonomics*, vol. 39, no. 4, pp. 621–627, 2009, special issue: Felicitating Colin G. Drury. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016981410800053X

[201] J. Nielsen and R. Molich, “Heuristic evaluation of user interfaces,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’90. New York, NY, USA: Association for Computing Machinery, 1990, p. 249–256. [Online]. Available: https://doi.org/10.1145/97243.97281

[202] J. Nielsen, “Enhancing the explanatory power of usability heuristics,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’94. New York, NY, USA: Association for Computing Machinery, 1994, p. 152–158. [Online]. Available: https://doi.org/10.1145/191666.191729

[203] J. Nielsen, *Heuristic evaluation.* John Wiley & Sons, 1994.

[204] J. Nielsen, “10 usability heuristics for user interface design,” *Neilsen Norman Group Website*, November 2020. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/

[205] D. Norman, *The Design of Everyday Things.* Basic Books, 1988.

[206] B. Shneiderman, *The Eight Golden Rules of Interface Design.* Pearson, 2016. [Online]. Available: https://www.cs.umd.edu/users/ben/goldenrules.html

[207] J. Surowiecki, *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations.* Doubleday, 2004.

[208] J. Nielsen and T. K. Landauer, “A mathematical model of the finding of usability problems,” in *Proceedings of ACM INTERCHI’93 Conference*, ser. INTERCHI ’93, April 1993, pp. 206–213.

[209] B. Neumann, “Bayesian Compositional Hierarchies - A Probabilistic Structure for Scene Interpretation,” in *Logic and Probability for Scene Interpretation*, ser. Dagstuhl Seminar Proceedings (DagSemProc), A. G. Cohn, D. C. Hogg, R. Möller, and B. Neumann, Eds., vol. 8091. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008, pp. 1–16. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2008/1605

[210] R. S. Wazlawick, *7.8: Organizational Hierarchy.* Elsevier Inc., 2014, ch. Chapter 7: Conceptual Modeling: Patterns.

[211] M. Philips, “Design principles - an introduction to visual hierarchy,” *Toptal: Designers Blog*, 2016. [Online]. Available: https://www.toptal.com/designers/visual/design-principles-hierarchy

[212] D. Raggett, A. L. Hors, and I. Jacobs, Eds., *HTML 4.01 Specification.* W3C, December 1999. [Online]. Available: https://www.w3.org/TR/html40/

[213] Wikipedia contributors, “Don’t repeat yourself,” *Wikipedia*, August 2021. [Online]. Available: https://en.wikipedia.org/wiki/Don’t_repeat_yourself

[214] PhantomJS contributors, “PhantomJS - scriptable headless browser.” [Online]. Available: https://phantomjs.org/

[215] Apple Inc., “Webkit: A fast, open source web browser engine.” [Online]. Available: https://webkit.org/

[216] M. Schuster, A. Domene, R. Vaidya, S. Arbanowski, S. M. Kim, J. W. Lee, and H. Lim, “Virtual device composition,” in *Eighth International Symposium on Autonomous Decentralized Systems (ISADS’07)*, 2007, pp. 270–278.

[217] M. Merabti, P. Fergus, O. Abuelma’atti, H. Yu, and C. Judice, “Managing distributed networked appliances in home networks,” *Proceedings of the IEEE*, vol. 96, no. 1, pp. 166–185, 2008.

[218] A. Owen and K. Martinez, “A dynamic hierarchical approach to modelling and orchestrating the web of things using the DOM, CSS and javascript,” in *CHI EA ’19: Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, May 2019. [Online]. Available: https://eprints.soton.ac.uk/432772/

[219] Arduino AG, “Arduino - home.” [Online]. Available: https://www.arduino.cc/

[220] OpenJS Foundation, “jQuery: Write less, do more.” [Online]. Available: https://jquery.com/

[221] Espressif Systems (Shanghai) Co., Ltd., “ESP8266 Wi-Fi MCU.” [Online]. Available: https://www.espressif.com/en/products/socs/esp8266

[222] T. A. Jr., Ed., *CSS Custom Properties for Cascading Variables Module Level 1*. W3C, December 2015. [Online]. Available: https://www.w3.org/TR/css-variables-1/

[223] R. Jota, A. Ng, P. Dietz, and D. Wigdor, “How fast is fast enough? a study of the effects of latency in direct-touch pointing tasks,” in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2013, pp. 2291–2300.

[224] A. Alabbas and J. Bell, Eds., *Indexed Database API 3.0*. W3C, August 2021. [Online]. Available: https://www.w3.org/TR/IndexedDB/

[225] reddit inc., “css_irl.” [Online]. Available: https://www.reddit.com/r/css_irl/

[226] Apple Inc., “Supporting dark mode in your interface,” *Apple Developer Documentation*, 2021. [Online]. Available: https://developer.apple.com/documentation/uikit/appearance_customization/supporting_dark_mode_in_your_interface

[227] Z. Kis, K. Nimura, and D. Peintner, Eds., *Web of Things (WoT) Scripting API*. W3C, September 2017. [Online]. Available: https://www.w3.org/TR/2017/WD-wot-scripting-api-20170914/

[228] Home Assistant, Inc., “Home assistant: Area registry.” [Online]. Available: https://developers.home-assistant.io/docs/area_registry_index

[229] The openHAB Community and the openHAB Foundation e.V., “openHAB: semantic model,” *openHAB Website*, 2021. [Online]. Available: https://www.openhab.org/docs/tutorial/model.html

[230] T. Anderson, “MySpace.” [Online]. Available: https://myspace.com/

[231] WordPress Foundation, “Blog tool, publishing platform, and CMS - wordpress.org.” [Online]. Available: https://wordpress.org/

[232] P. Cowburn, Ed., *PHP: PHP Manual.* The PHP Documentation Group, October 2021. [Online]. Available: https://www.php.net/manual/en/

[233] UK Health Security Agency, *COVID-19: guidance for maintaining services within health and care settings - infection prevention and control recommendations.* UK Health Security Agency, September 2021, Available: https://www.gov.uk/government/publications/wuhan-novel-coronavirus-infection-prevention-and-control/covid-19-guidance-for-maintaining-services-within-health-and-care-settings-infection-prevention-and-control-recommendations.

[234] Google LLC, “Lit: Simple. fast. web components.” [Online]. Available: https://lit.dev/

[235] J. Diggs, S. McCarron, M. Cooper, R. Schwerdtfeger, and J. Craig, Eds., *Accessible Rich Internet Applications (WAI-ARIA) 1.1.* W3C, December 2017. [Online]. Available: https://www.w3.org/TR/wai-aria/

[236] Wikipedia contributors, “Create, read, update and delete,” *Wikipedia*, October 2021. [Online]. Available: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

[237] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *RFC-2616: Method Definitions.* W3C, June 1999. [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[238] J. Postel, *RFC-768: User Datagram Protocol.* IETF, August 1980. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc768