

# Real-time, distributed, and collaborative user interfaces applied in digital workspaces

*by*  
*JESSE CLAVEN*



THE UNIVERSITY OF QUEENSLAND

School of Information Technology and Electrical Engineering,  
The University of Queensland.

Submitted for the degree of  
Bachelor of Engineering  
in the field of Software

JUNE & 2015.



Jesse Claven  
41000816  
1/19 Ellis St.  
Kangaroo Point

June 15, 2015

Prof Paul Strooper  
Head of School  
School of Information Technology and Electrical Engineering  
The University of Queensland  
St Lucia, Q 4072

Dear Professor Strooper,

In accordance with the requirements of the degree of Bachelor of Engineering in the division of Software Engineering, I present the following thesis entitled:

“Real-time, distributed, and collaborative user interfaces applied in digital workspaces”.

This work was performed under the supervision of Dr Mark Schulz. I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

JESSE CLAVEN.



# Acknowledgments

I would like to thank my supervisor, Dr Mark Schulz, for his constant guiding hand in keeping me focused on tackling a project with real world application. Also, for his technical direction in using Meteor and help with planning the prototype example.



# Abstract

Traditionally user interfaces have been designed for a single user using one common device type—e.g. someone on a computer visiting a website. With the internet and mobile devices now being commonplace, interfaces could take advantage of being distributed across devices and working collaboratively with others in real-time. While there have been attempts to handle this (e.g. Google Docs), they have so far been in a limited, prescribed manner. A proposed concept is put forward to design and build a new approach for a distributed and real-time collaborative user interface focusing on the concept of having a workspace with components that the user is able to freely use in a real-time manner. It is based upon existing web browsers and devices. Parts of the UI can be distributed across separate platforms. A prototype of a workspace for education is included and user testing of the prototype shows positive experiences and results for the users.

# Contents

<b>Acknowledgments</b>	<b>v</b>
<b>Abstract</b>	<b>vii</b>
<b>List of figures</b>	<b>x</b>
<b>List of code samples</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Current progression of user interfaces . . . . .	1
1.2 Proposal . . . . .	3
1.3 Overview . . . . .	4
<b>2 Literature review and prior art</b>	<b>6</b>
2.1 Real-time . . . . .	6
2.2 Collaborative . . . . .	7
2.3 Distributed User Interface . . . . .	11
<b>3 Concept, prototype and example</b>	<b>14</b>
3.1 Concept . . . . .	14
3.1.1 Workspace . . . . .	14
3.1.2 Canvas . . . . .	16
3.1.3 Component . . . . .	17
3.1.4 Users and roles . . . . .	20
3.1.5 Interface . . . . .	20
3.1.6 Communication . . . . .	21
3.2 Prototype . . . . .	22
3.2.1 Architecture . . . . .	22
3.2.2 Templates . . . . .	23
3.2.3 Schemas . . . . .	24
3.2.4 Usable computers and devices . . . . .	24
3.2.5 Users . . . . .	24



3.2.6	Data protocol . . . . .	25
3.2.7	Real-time . . . . .	26
3.2.8	Data storage . . . . .	27
3.2.9	Chat communication . . . . .	28
3.3	Example workspace: PWM/Colour theory . . . . .	28
<b>4</b>	<b>User testing and results</b>	<b>33</b>
4.1	User testing . . . . .	33
4.1.1	Distance education use . . . . .	33
4.1.2	In-class use . . . . .	35
4.1.3	Out-of-class use . . . . .	36
4.2	Results . . . . .	36
<b>5</b>	<b>Conclusion and future work</b>	<b>37</b>
	<b>Appendices</b>	<b>39</b>
<b>A</b>	<b>Prototype</b>	<b>40</b>
<b>B</b>	<b>Example</b>	<b>43</b>
	<b>Bibliography</b>	<b>45</b>

# List of Figures

2.1	Johansen's CSCW time/space matrix . . . . .	8
3.1	Server index of available workspaces and rooms . . . . .	16
3.2	Server index of available workspaces and rooms . . . . .	29
3.3	Chat room with users and messages . . . . .	30
3.4	Example canvas . . . . .	31
3.5	DUI example: single component on an iPhone . . . . .	32
B.1	Electronics schematic . . . . .	43

# Code samples

3.1	Workspace schema . . . . .	15
3.2	Canvas schema . . . . .	16
3.3	Object schema . . . . .	18
3.4	Data source schema . . . . .	19
3.5	User role schema . . . . .	20
A.1	PWM input component . . . . .	40
A.2	PWM input component . . . . .	41



# Chapter 1

## Introduction

### 1.1 Current progression of user interfaces

The user interface has improved greatly over time, starting from simple command-line interfaces to high-resolution, multi-touch. In tandem, computing has gone from handling one task at a time to multi-tasking, and from isolated machines to reliance on networks and the internet. The internet is what enables most modern UI concepts such as being collaborative, real-time, and distributed. There has been a shift from users working by themselves at their workstations to having their work live in the ‘cloud’ and being able to easily invite others to collaborate. For example, both Microsoft and Apple have shifted their respective suite of office programs to having cloud-based versions that run in the browser. [1] [2] The McKinsey Global Institute in their research on ‘Global flows in a digital age’ detailed record a ‘\$26 trillion flow of goods, services, and finances in 2012’ with an expected \$85 trillion by 2025. [3] The next focus for UIs is moving to those that are real-time and distributed, while being collaborative.

With networks and the transfer of work from pen and paper to computers came computer-supported cooperative work (CSCW), also known as groupware. CSCW focuses on technology that enables people to work in groups and includes software, networking, and hardware. Other areas such as education, video games, and communication have also had an increased in the number of users working together. Adding to the example of online office suites though that’s only a common example though many other types of work are now online such as video creation (YouTube) and programming (Cloud9). These services are revolutionising industries such as training. [4] So far though with all these applications they’re constrained to specific, single activities and do not give users freedom to mix related activities and components together like they would outside of the digital world. A primary example related to education are the remote laboratories run by the Centre for Educational Innova-

tion and Technology (CEIT) at The University of Queensland where student's main interaction is inputting experiment values on a page by them self.

Low-latency networks allow for real-time events with real-time events (running at the same rate as in the non-digital world). For example, when editing a document the changes show immediately for all users viewing the document. While real-time software is beneficial, it introduces technical challenges such as how to handle conflicts in changes, control over UI elements, and integrity of data. Real-time is not only for within an application but can also be used to interact with outside sources. The most well-known applications are Google Drive for document editing and Facebook for notifications and commenting.

A distributed user interface (DUI) is one which is not centered on a single user one one platform but can be divided and be available on other platforms such as through the internet or on different devices and environments. DUI are also dependent on the task, affording multiple users to be involved. [5] As the internet has enabled a more connected society across the world, communication and cooperative involvement has increased. This has caused for the user interfaces and user experiences to be redesigned to appropriately handle this new usage paradigm. DUIs are still simple and focus on multiple screens connected to a single device such floating windows in an application. DUIs are relevant to collaborative scenarios in that, "Under a collaborative scenario, users sharing common goals may take advantage of DUIs to carry out their tasks because they give a shared environment where users are allowed to manipulate information in the same space at the same time. Under this hypothesis, collaborative DUI scenarios open new challenges to usability evaluation techniques and methods." [5]

Mark Weiser wrote the seminal paper, 'The Computer for the 21st Century' where he envisioned a future where technology (specifically, computers) "weave themselves into the fabric of everyday life until they are indistinguishable from it"—commonly now referred to as ubiquitous computing. Computers and mobile devices are distinct objects, a 'world of their own', and that 'the idea of a "personal" computer itself is misplaced and that the vision of laptop machines, dynabooks and "knowledge navigators" is only a transitional step toward achieving the real potential of information technology. Such machines cannot truly make computing an integral, invisible part of people's lives. We are trying to conceive a new way of thinking about computers, one that takes into account the human world and allows the computers themselves to vanish into the background.' [6] Integral to this vision is the user interface and moving beyond it being a concrete, incorrigible element on one screen and platform.

## 1.2 Proposal

While we are still away from Weiser's prescient vision we can take advantage of the current state of technology to continue the UI progression.

I propose a framework for allowing users to collaboratively connect components together and manipulate them within a canvas in a real-time workspace with the UI elements in the workspace being able to be divided out by themselves in a distributed manner onto other platforms, entirely separate from the main workspace UI.

A workspace would be an analogous digital representation of a workspace you find at an office desk, classroom, laboratory, or anywhere. Within this they wouldn't be constrained to specific activities, but would be able to do any type of activity that can be represented with components and actions within the workspace. The workspace, and everything within it, would be represented with schemas which allows for easy creation and modification, and also for different technologies to be used in implementation. A benefit of schemas and not being bound to a specific implementation is that the concept can be implemented in newer technologies. Another notable part is that every schema has an associated template which contains the structure, style, and actions (events) for that schema.

Workspaces will consist of definable modules. Predominately these will be canvases, components, and communication channels. Canvases are comparable to an art canvas or empty desk in that it's a slate where work is done. Components include, but aren't limited to, objects which can be entirely abstract or represent an object from the real world and have definable ways in which they can be interacted with and manipulated, and data sources which are connections to data within the workspace (i.e. another object) or externally (i.e. a sensor).

The framework would be browser-based which means almost any modern device in the last decade to be utilised. Web browsers have become quite advanced beyond simple hypertext elements and now contain rich APIs for storage, 3D rendering, state handling, etc. JavaScript especially has matured as a language and has different libraries available to be used allowing for a diverse type of components and actions within a workspace. These workspaces being available through the internet would allow for easy collaboration for people wherever they have internet access and a platform with a browser.

The modular and responsive nature of web design means that UI elements can be divided and distributed to separate platforms such as a phone.

Workspaces would rely on streaming connections to have real-time interaction for users within amongst all components in the workspace, and also externally connected.

This concept of a workspace is important as it doesn't rely on emerging technol-

ogy but current, proven ones. It has low technological requirements so that people from diverse background and socio-economic situations would be able to use it. Quick prototyping and experimentation is possible due to not being solely reliant on physical components. Also, it moves DUIs from being bespoke ones, dependent upon specific use cases and technology, to more easily created and used.

With Web 2.0 there was an influx of user-created content—instead of simply going to websites and consuming what was available, they became creators. This workspace would take that further with freeing them from the specific web-applications for content creation by giving them a workshop with tools and objects to create with.

The workspace isn't entirely separated from the physical world either. While components may be visually represented in the workspace, they can be connected to the real physical component. For example, a class working on a remote—or even local—laboratory may have data being output from a sensor that they wish to see and manipulate; these would be data sources. Another use case is when at an event there could be sensors that detect attendees and interact with their devices collaboratively. This would be beneficial at university lectures where students have needed to either open URLs or register physical clickers to be involved.

Frank Chimero, a prominent designer and writer, spoke of designing for print as more of an artistic approach as once it's printed, that's forever how it is. With designing for the web, he saw it as more of 'joining' non-uniform objects together. This is the inherent structure of workspaces. The workspace described above is like his description of "an edgeless surface of unknown proportions comprised of small, individual, and variable elements from multiple vantages assembled into a readable whole that documents a moment". [7] The described workspace would be one of the first attempts at this.

## 1.3 Overview

The workspace concept will be explored in further detail along with a prototype of the workspace being created. User testing will be done to evaluate the relevance and usefulness of the concept. The prototype will contain an example workspace representative of teaching pulse width modulation and colour theory to students and will be connected to a physical RGB LED and colour sensor to demonstrate the workspace being connected to external objects. This was chosen due to the focus of CEIT and the prevalence of online learning.

With web applications there are many technical details to consider such as security, scalability, performance, etc. These will not explicitly be covered in this thesis though it will be shown how they're inherently covered in the choice of design,



architecture, and technology.

# Chapter 2

## Literature review and prior art

### 2.1 Real-time

With the web the current applications of real-time are primarily streaming of media (audio/video), content updates (e.g. social media feeds), and text editing (for writing and coding). There are other less-common use cases such as wireframing (Precursor) [8] and communication (Google Wave). [9]

For handling edit conflicts there are naïve and advanced approaches, most of which rely on a client-server model. Naïve approaches consist of preferring one user's edits over another or the use of a lock though one would argue that a lock removes real-time ability. The first successful collaborative real-time text editor was Etherpad which launched in 2008. It was acquired by Google a year later and released as open source. It features a synchronisation model that is viewed as an early, informal version of operational transform. [10]

Operational Transform is a class of algorithms for handling real-time, concurrent transactions such as edits in a document. Local replication sets are kept of shared data and then operations are transformed and applied. There has been much research and implementation of various algorithms in the last two decades that vary in performance, consistency, and data types. For example, Chengzheng and Ellis have reviewed four such approaches and proposed one of their own. Even with these multiple algorithms there are still unexplored issues such as operation granularity and due to the lack of OT being used in real-life collaborative environments there lacks data that could be used to further OT. [11] Randolph, Boucheneb, Imine, et al. have also done research into the consistency of OT and found that using the position and character parameters in transactions are necessary, they're not sufficient to ensure consistency.

There exists 2 main real-time web frameworks, Meteor [12] and DerbyJS. [13] Both are model-view-controller frameworks which includes templating, data storage,

and reactive models. While both are full-stack frameworks, Meteor is more expansive in its packaging system, development tools, and plugin ecosystem. DerbyJS though is based upon the ShareJS implementation of OT. [14] DerbyJS also server-side renders pages by default which while beneficial for the initial page load can use bandwidth by sending full HTML as opposed to simple data changes to the client, where the client itself then re-renders the affected parts of the page.

There are also real-time database solutions such as Firebase [15] and Simperium [16] but their benefits of backups and performance aren't important to the overall concept presented in this thesis.

## 2.2 Collaborative

Collaboration is constrained by periods of time. For example, people can collaborate by post and are constrained by postal delivery times. Different types of collaboration prefer, and even require, certain periods of time. That collaboration can be beneficial is known. Technology allows for different methods and time periods for collaboration and is often defined under the computer-supported cooperative work term. Johansen in 1988 defined a matrix (see figure 2.1) [17] for the types of collaboration that happens with CSCW as far as the time and space are considered. [18] We'll be focusing on same time and different place.

On real-time software, Gutwin, Greeberg, and Roseman said "In real-time groupware systems that provide a shared virtual space for collaboration, the possibilities for interaction are impoverished when compared with their physical counterparts." [19] Little advancements have been made in 9 years. There's the previously mentioned Google Drive and the Precursor applications which excel in their specific tasks, there's no real-world examples of real-time systems that are a 'shared virtual space for collaboration'. Now that progression has been made in foundational parts of groupware (e.g. networking, devices, etc.) it is important to reconsider a modern, real-time groupware system which is what we'll be doing in this thesis.

For cooperation to succeed there needs to be mutual flexibility and conflict prevention. [20] Allwood listed the following goals for users to try and maintain to ensure this:

- Mutual friendliness
- Lack of tension (tension release)
- Lack of need to defend a position
- Admitting weakness or uncertainty

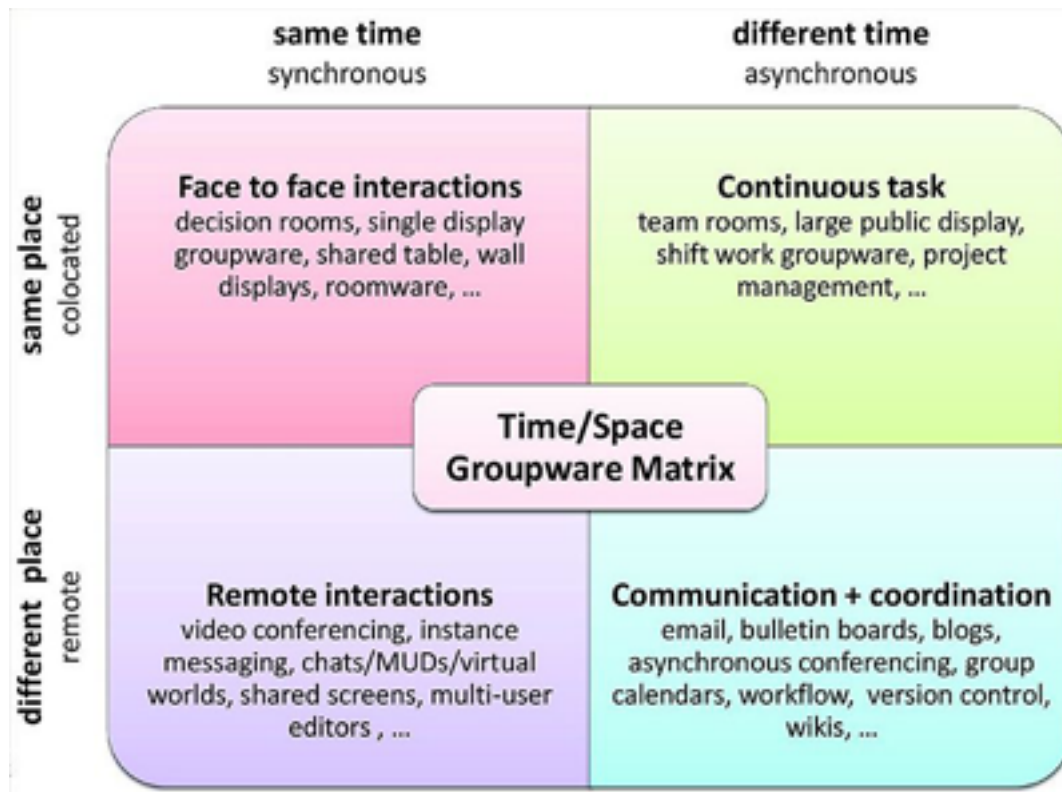


Figure 2.1: Johansen's CSCW time/space matrix

- Lack of attempts to overtly impose opinions on others
- Coordination of attention and movements
- Giving and eliciting feedback expressing mutual support and agreement
- Showing consideration and interest
- Invoking mutual awareness and beliefs

Through different verbal and nonverbal means mutual flexibility and conflict prevention can be found. [20] Verbal can also include written communication. Doing this can be hard due to text not clearly delivering the emotional intent due to the lack of verbal cues. With nonverbal, aside from webcams there are little means for showing this and for responding.

Gutwin, Greenberg, and Roseman refer to 'workspace awareness', where "The rich person-to-person interaction afforded by shared physical workspaces allows people to maintain up-to-the minute knowledge about others interaction with the task environment. ... part of the glue that allows groups to collaborate effectively." [19] In their work from 1996 they use a 'What You See Is What I See' (WYSIWIS) model that relies on having a widget for each user which shows a scaled down view of that person's workspace and for 'What You See Is What I Do' (WYSIWID) they have

a widget which shows the immediate context around the other user's cursor. They also define 'Workspace Teleportals' which allow a user to see another user's entire screen. They found the benefits of workspace awareness to be:

- Direct communication. People explicitly communicate information about their interaction with the workspace; this communication is primarily verbal, although gestures (Tang, 1991) and deictic references (Tatar et al., 1991) are also common.
- Indirect productions. People commonly communicate through actions, expressions, or speech that is not explicitly directed at the other members of the group, but that is intentionally public (Dourish & Bellotti, 1992; Heath & Luff, 1991).
- Consequential communication. Watching or listening to others as they work provides people with a great deal of information about their interaction with the workspace (Segal, 1994).
- Feedthrough. Information can also be gathered by observing the effects of someones actions on the artifacts in the workspace (Dix et al., 1993).
- Environmental feedback. People also perceive higher-level feedthrough from the indirect effects of anothers actions in the larger workspace. For example, in a control room situation, seeing some measured value decrease can provide evidence that another member of the team has initiated a particular procedure.

After testing their widgets they found that users did not like the WYSIWID possibly due to it not having smooth performance and the widget fit not fitting well into the overall interface layout.

With the rising popularity of online education, "Beyond the classroom, new technologies are providing more and more tools for users (Moodle, Atutor, Blackboard, Calorina, etc.). However, these technological systems do not specifically support education, and learning is not a key element in design and development. Instead, these models are based on the advantages teachers and students can find in managing online and distance education." [5] These applications though are a realisation of Bouras, Lampsas, Zarafidis et al. suggested future work from 1998. [21]

Gerber, Grundt, and Grote [22] found that student's performance in online, collaborative environments, was impacted by their content and inter-personal communication with other students, and with the involvement of the tutors. Previous studies had focused on the importance of quantity of communication as opposed to

quality (content). While it's extremely common for CSCW software to have communication functionality, usually in the form of chat or forums, and they're usually separated from the content or at most are a simple commenting system on files. While commenting on files is content orientated communication, it's static viewing as opposed to having a dynamic object that can be explored, connected with other objects, etc. which limits how the object (file) can be understood and discussed.

With Massive open online course (MOOCs) the courses sometimes have communication in the forms mentioned above but it is not integrated with the course content itself.

"From a conceptual perspective, one would hypothesize that there are significant relationships between the amount of interaction and student learning process and performance. The positive relationship between interaction and learning performance has been clearly documented in traditional classroom settings (Powers & Rossman 1985). In web-based distance learning, a number of studies showed that students typically report increased learning (Jing&Ting 2000;Swan et al. 2000; 2001; Picciano 2002; Rovai&Barnum 2003) and satisfaction (Jing & Ting 2000; Swan et al. 2000; Hong 2002; Richardson & Swan 2003), depending on the quantity of interactions with peers and the instructor." [22]

A traditional, instructionist approach to learning in a classroom is focused on delivering strict instructions to students and for them to execute them in an expected way. While some maintain that this approach is good for fundamental skills [23], it can be ineffective as "if their initial understanding is not engaged, they may fail to grasp the new concepts and information that are taught, or they may learn them for purposes of a test but revert to their preconceptions outside the classroom." [24].

MOOCs have carried over this instructionist approach. An alternative to the instructionist approach is Papert's constructionism which is focused on 'learning by making' where learning is treated as 'building knowledge structures' instead of forced ways of understanding knowledge [25]. There are various metacognitive techniques that help students take control of their own learning by defining their own goals, and by the teachers monitoring the students in their progress, as opposed to a student being directed in what the goals should be for their learning activities [24]. Papert does not claim that this the absolute best method for learning but that we should be more flexible in how students are allowed to learn. He does argue that the constructionism approach is more inclusive of the range of learning styles. His experience from observing an art class details this well:

".. the art room I used to pass on the way. For a while, I dropped in periodically to watch students working on soap sculptures and mused about ways in which this was not like a math class. In the math class students are generally given little problems which they solve or don't

solve pretty well on the fly. In this particular art class they were all carving soap, but what each students carved came from wherever fancy is bred and the project was not done and dropped but continued for many weeks. It allowed time to think, to dream, to gaze, to get a new idea and try it and drop it or persist, time to talk, to see other people's work and their reaction to yours—not unlike mathematics as it is for the mathematician, but quite unlike math as it is in junior high school.”

CEIT runs iLabs at UQ that enables real laboratories to be accessed through the internet. They have several experiments on nuclear radiation, an inverted pendulum, and a number of experiments in power engineering utilising AC and DC motor-generator pairs. [26] Students are given access to a web interface where they can enter input values related to the laboratories and receive the results. There's no direct manipulation with the laboratories themselves. This effective in enabling students from all over the world to have access to expensive equipment and experiments that they otherwise would not be able to experience and learn from.

Another approach to laboratories is to simulate the hardware through software. Ma and Nickerson found “that students learn not only from equipment, but from interactions with peers and teachers. New technologies may call for new forms of coordination to augment or compensate for the potential isolation of students engaged in remote learning.” [27]

This thesis joins the two approaches in that while the workspace can be seen as a simulation, it can also be directly connected to real-world equipment, as will be seen in the prototype example. This meets one of the suggested areas of research by Ma and Nickerson in, “Perhaps with the proper mix of technologies we can find solutions that meet the economic constraints of laboratories by using simulations and remote labs to reinforce conceptual understanding, while at the same time providing enough open-ended interaction to teach design. Our review suggests that there is room for research that seeks to create such a mix, which might be informed by studies of coordination as well as the interactions that lead students to a sense of immersion.” [27] Workspaces need not be limited to science and engineering applications only either but also for art, design, programming, and anything that can be designed and programmed within a web browser.

## 2.3 Distributed User Interface

Villanueva, Tesoriero, and Gallud, devised a classification scheme for DUIs where they can be classified as divisible and distributable. [5] They use Paint.net and GIMP as case studies where you can separate floating toolbars from the main application frame with this being classed as divisible only because while you have

the separate toolbars though it is not distributive, being only one platform. Distributable was reserved for UI objects that could be moved to entirely different platforms. This is an important distinction to make because it moves DUI from being simply splitting UI elements apart, to truly distributing them to other platforms (i.e. other devices, interaction mediums, etc.). Within their model the UI system also has various states: unified where there is at least one UI system that is not dependent on another and distributed if there are at least two UI systems that are on different platforms.

Radle, Jetter, and Reiterer found that when designing a DUI especially for web searching as a group (TwisterSearch), the participants felt that they were able to contribute more fairly, they were able to use TwisterSearch in novel, unexpected ways (as a mind map tool), and allowed for different ways to approach the presented problems. [28]

Video games have often led the way in innovative and new technology. The video game Supreme Commander allowed players to use a secondary screen to display the map [29], the Xbox One console has a companion app on phones called SmartGlass that has controls for the console and shows secondary information [30], and the Wii U console's controller has a small, built-in screen for similar purposes. [31]

With Apple's iOS 8 and Mac OS Yosemite, released in 2014, they introduced a feature known as Continuity which allows Macs and iOS devices to work together, though not specifically in a distributed way as far as dividing UI elements across devices, but focused on continuing activities across devices. For example, if you're writing a Message on your iPhone you're able to continue doing so on your Mac, or you can answer phone calls through your Mac. [32]

A related concept to DUIs is that of 'context-aware computing' which "appeared along with mobile platforms to adapt the applications to these new more restricted devices. In these contexts the user was alone on the platform but could switch from one platform to another (in general only one at the same time)." [5] The above Continuity feature could be considered an application of this. With reportedly the majority of US cell phone owners using their mobile devices to access the internet in 2014 [33], web design was in need of design principles for handling the diversity of screen sizes. Ethan Marcotte published an article about 'Responsive Web Design' where he details the principle of a design 'responding' to the device and changing the design appropriately. [34] This, tied with context-aware computing, is fundamental for truly distributed user interfaces for if you distribute a UI element without considering the hardware aspects of its destination it might be entirely unusable.

Responsive design requires a designer to thoughtfully design each response. User interface generation is a way in which this could be automated using algorithms. Tran, Vanderdonckt, Kolp, et al. defined a process to generate UIs for database



applications from declarative task, user, and domain models. [35] The task is the abstract UI, the domain the features of the objects within the UI, and user the preference of the users. Their work is an important step in UI generation and focuses on creating UIs from a declarative manner as opposed to procedural, from a designer's workload. Similar work was done by Penalver, Lazcorreta, and Lopez, though using schemas as the foundation for the generated UI. [36]

In 'Distributed User Interfaces: Collaboration and Usability' Tesoriero and Lozano reviewed DUIs from the point of view of usability and collaboration applications and found that: "It is common to see a user employing in his/her daily work three different computing systems such as a mobile or smartphone, a desktop computer and a tablet. However, the DUI scenario is not limited to single user interaction." [5]

An important area of DUIs is security and privacy which has been considered with different concepts such as 'proof based access control' [5] but these will not be considered in this thesis.

Tabletops such as Microsoft PixelSense [37], TwisterSearch, and TangiSense [5] are one of the most common applications of DUIs. TwisterSearch uses iPads for users to enter search terms that are then display on the central tabletop and TangiSense uses physical objects with RFID chips. Both are limited their application of DUIs principles. TwisterSearch is not divisible and is distributable only in the sense that the task is completed on multiple platforms which each have a distinct UI as opposed to a master UI being distributed. TangiSenses cover the issue of when there is a single master UI and then subsets of that UI on the other environments though whether it is an 'issue' is dependent upon the purpose of the UI as for some tasks the master-subset would be an ideal design pattern, not an issue. TangiSense is collaborative in that multiple tabletops can be connected together, and distributed in that multiple platforms (such as tablets) can be used.

In all the examples presented in 'Distributed User Interfaces: Collaboration and Usability' they were all bespoke applications with specific but limited possible divisibility and distributability. While some of the applications used the internet for the tasks, such as TwisterSearch [28] and CSchool [38], they did not take advantage of it for distributing the UI itself, but rather separating completing the task into separate UIs. [38] Kwon, Lee, and Musyaffa describe a DUI for a web-based application for collaborative social authoring though it has no real divisible or distributable UI elements and is more of a context-aware/responsive UI—for example, they speak of how phones have smaller screens and therefore less space for UI elements which limits the media authoring they can do. It is able to distribute the task, in a way, by allowing it to be worked on more than one platform.

# Chapter 3

## Concept, prototype and example

The workspace concept will be more fully explored and detailed, along with including sample schemas—the properties shown in the code samples are not the only possible properties. The feasibility and effectiveness of the proposal will be shown by the simplicity of it and the ease of implementation shown later in the prototype and example. Most implementation details will be covered in the prototype section.

### 3.1 Concept

#### 3.1.1 Workspace

An abstract workspace is an edgeless container of no specific size. There is complete freedom in what can be contained in a workspace though there will be conventional defaults. This is where the analogy to a real world desk shows in that like that desk, you can place anything on it, and what’s on it is generally dependent upon the task at hand. An example schema can be seen in code sample 3.1. Similar to the real world again, the workspace is multimodal allowing for multiple types of input devices.

A room—or, concrete workspace—is an instance of a workspace that has been created from the schema and contains instances of the components for that workspace. Rooms currently are separate entities from each other. The lifetime of a room is dependent upon the users. Where they’re stored depends on the architecture and with the following prototype we’ll be using a standard client-server architecture. The creation of an instance is an implementation detail.

The rendered workspace (room) is the user interface, including user interaction/-experience, and are dependent the things within the workspace such as the canvas and components. Within the rendered workspace is where all activity is contained such as component manipulation. It’s once rendered that the workspace has a size which usually will be the size of the screen of the device where the workspace has

been rendered. See figure 3.2 for a screenshot of an example workspace.

#### Code samples 3.1: Workspace schema

```
name: {
  type: String,
  label: 'Workspace_name'
},
description: {
  type: String,
  label: 'Supported_workspace_purposes_and_activities'
},
components: {
  objects: {
    type: [String],
    label: 'Array_of_object_IDs'
  },
  dataSources: {
    type: [String],
    label: 'Array_of_data_source_IDs'
  },
},
communication: {
  channel: {
    type: Boolean,
    label: 'Communication_channel_enabled'
  }
},
template: {
  type: String,
  label: 'Template_used_for_rendering'
},
requireAuth: {
  type: Boolean,
  label: 'Is_authentication_required_for_this_workspace?'
},
maxUsers: {
  type: Integer
  label: 'Max_number_of_users_that_can_be_in_this_workspace'
}
```

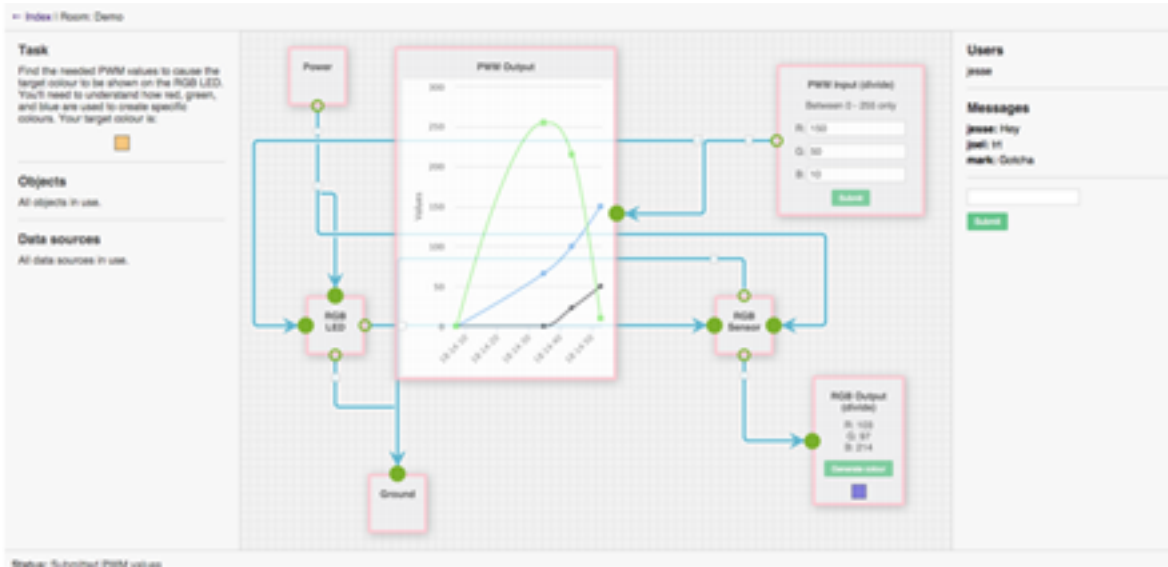


Figure 3.1: Server index of available workspaces and rooms

### 3.1.2 Canvas

The abstract canvas is an item within the workspace which similarly is edgeless and without specific size by definition, with only one canvas per workspace for now. The dimensionality of a canvas is 2D and cartesian in use of x, y coordinates. While in the real world a canvas, or area on a desk, would be of a certain material such as paper or wood, the analogous digital concept is not considered in this thesis.

When rendered, a canvas is similar to a workspace in that it takes on a size generally dependent on the size of the screen, and how much of the screen is for the canvas. It has boundaries that the components cannot move outside of.

All actions on the canvas is shown in real-time across all devices for the users in that room. Components are placed on the canvas and can be dragged and dropped to change position. Depending on the workspace, components can either be in a sidebar and then dragged onto the canvas, or they can have initial coordinates where they will be placed when a workspace instance is made.

#### Code samples 3.2: Canvas schema

```
name: {
  type: String,
  label: 'Workspace_name'
},
description: {
  type: String,
  label: 'Supported_workspace_purposes_and_activities'
```

```

},
background: {
},
draggable: {
  type: Boolean,
  label: 'Are components draggable?'
},
resizes: {
  type: Boolean,
  label: 'Can the canvas be resized by the user?'
},
template: {
  type: String,
  label: 'Template used for rendering'
}

```

### 3.1.3 Component

Component instances can be connected together with visual flow lines. These connections are to anchor points on the components which are either for input or output. What is sent along these flow lines are messages of any type of data. For example, a power source component would send its current/voltage as an output represented with integers. This is a simple implementation of flow-based programming which is a simple programming concept that connects ‘black boxes’ which send data across connections. [39] This is an easy and intuitive approach for all users to be able to use the workspace, as opposed to complex UI, or requiring them to specify textually the connections.

What the output is from a component can be continuous or discrete values, or a definable function in the schema. Inputs are handle via function as well attached to the input connector in the schema. There can be transform functions applied to objects as well. For example, a simple transform function could be to scale an input value by half. All function types would be defined in the implementation’s programming language.

### Object

One of the main types of components are objects. Objects can be anything that can be defined in a schema and rendered and manipulated within a browser. Usually these will represent objects from the real world (such as the power source example) though they can be more abstract things that may not necessarily exist physically. They can be connected to the real world representation by use of data sources. For

example, there could be a graph object which represents the weather which would then have an input connection to a weather data source which in turn is connected to a weather monitoring station.

Skeuomorphism, where the design retains a ‘realistic’ look over the object it represents such as a notebook application looking like paper, would not necessarily be needed or recommended. As Apple and other companies found, they were beneficial when software was first becoming adopted and is now unnecessary. While the metaphor of physical workspace is used, a skeumorphic design wasn’t used as it restricts and constrains what is inherently digital to real-world objects.

#### Code samples 3.3: Object schema

```
name: {
  type: String,
  label: 'Workspace_name'
},
description: {
  type: String,
  label: 'Supported_workspace_purposes_and_activities'
},
input: {
  type: String,
  label: 'Function_(JavaScript)'
},
output: {
  type: String,
  label: 'Function_(JavaScript)'
},
template: {
  type: String,
  label: 'Template_used_for_rendering'
},
position: {
  type: String,
  label: 'x,y_value'
}
```

#### Data source

Data sources are focused on two-way message passing outside of the workspace. This enables the workspace to be used for many different activities and not restricted purely to the objects and the input and output functions that can be defined. Data

sources inherit the object schema as they too need to be rendered and interacted with. They differ in that they have a connection type specified, along with any needed parameters like a source URL.

Connections to these outside sources are possible using the internet and JavaScript. This allows for accessing WebSockets, REST, and anything that can be done with JavaScript, or anything that can be passed along as messages on those connection types. This will also be dependent on the implementation and what message protocols it can handle such as between MQTT or DDP, for example.

Data sources can be used to send commands in messages to the connected physical object. There are therefore two types of data sources, soft, being one that is not connected to a physical object, and hard, one that is—these can be anything that can receive messages, using the DDP protocol, through the internet. With the Internet of Things and how prevalent it is to have connected objects, this allows for a range of connections.

#### Code samples 3.4: Data source schema

```
name: {
  type: String,
  label: 'Workspace_name'
},
description: {
  type: String,
  label: 'Supported_workspace_purposes_and_activities'
},
inherit: {
  type: String,
  label: 'Schema_ID_that_this_is_based_on'
},
connectionType: {
  type: String,
  label: 'Connection_type_used'
},
source: {
  type: String,
  label: 'Source_path'
},
value: {
  type: String,
  label: 'Output_value'
}
```

### 3.1.4 Users and roles

The standard user is a simple participant. They can manipulate the workspace and work together with others. Different user roles can be created to suit the needs of the workspace. These roles can have greater, or lesser, abilities such as being able to take over control of a component that is in use by another participant. These abilities are defined by a schema by specifying allowed and disallowed actions. This allows for great flexibility in the types of users, and groups, that can use workspaces as in the real-world roles play a key part in group interactions.

Code samples 3.5: User role schema

```
name: {
  type: String,
  label: 'Name_of_role'
},
allowed: {
  type: [String],
  label: 'Array_of_allowed_actions'
},
disallowed: {
  type: [String],
  label: 'Array_of_disallowed_actions'
}
```

### 3.1.5 Interface

#### Distributed

The workspace being edgeless, components can be thought of as existing inside the canvas and floating outside the canvas. This second state allows components to be divided and distributed on other devices and environments. Components can exist in both states simultaneously in that they are rendered both on the main canvas and rendered by itself elsewhere as a distributed UI element. This allows real-time collaboration to continue.

This is achieved by the server having URLs for each component that can be accessed directly or by a user clicking on a 'divide' button that creates another window with just that component. Using responsive design it will be designed to take advantage of different screen sizes and different interaction methods.

This approach for a DUI is good because for users, their first experience is with the whole workspace as a master UI and from there, they themselves separate out certain parts of the UI, and in doing this it keeps the context in mind of where that



part came from, how it's connected, and so on.

### **Collaborative**

The workspace awareness concepts by Gutwin, Greenberg, and Roseman, while good, had various shortcomings due to the nature of the technology at the time. The workspace uses both 'What You See Is What I See' and 'What You See Is What I Do' in that as all users are seeing the components being moved and interacted with in real-time.

### **Real-time**

All interactions such as dragging components, entering data into input fields, etc. will be shown in real-time on all user's devices. Two main concerns are conflict resolution and latency. Currently most internet connections (both mobile through 3G and 4G and through broadband) are sufficient for large amounts of interactions to be shown near instantly. When there is high latency though a technique known as latency compensation can be used. When a user does something, instead of waiting to send this action to the server and receive a result, the action's effect is shown immediately on their device so as to appear as if there is no latency, while it is being sent to the server and the other user's in the workspace. This then falls back onto conflict resolution though with most group dynamics, with coordination only one user will be explicitly using one component at a time. A naïve mutual exclusion lock implementation can be used to lock components to users such as while dragging.

A related programming paradigm to real-time data is reactive programming which can be considered as 'programming with asynchronous data streams.' [40] Streams can be any sort of data from events to variables. Streams emit events and anything that is 'listening' to the stream acts once it receives an emitted event. It's similar to event-driven programming though the listening is done by subscribing to the stream, and with event-driven, they can be once-off events as opposed to a stream.

#### **3.1.6 Communication**

Communication, both verbal and nonverbal, is key for collaboration to succeed. [20] Communication channels can be used in workspaces such as text, audio, and video. While text has low performance issues, some users may not be comfortable with their typing speed or spelling ability. While audio and video have the benefits of easier communication they carry much higher bandwidth and processing requirements.

## 3.2 Prototype

I built a conceptual framework for the concept including core elements such as workspaces, canvases, components and communication (text only) were built. The method for building was to research available APIs and libraries, if any, and use those to build separate examples. These were then combined into one overall framework. This shows the detailed use of schemas and not being bound to specific implementations.

### 3.2.1 Architecture

A standard client-server architecture was used with Meteor, an open source JavaScript app platform, as the platform. Meteor is full-stack platform for building web applications in pure JavaScript. [41] As covered in the literature review (section 2.1) Meteor is the best choice as it was built around real-time capabilities including reactive programming which is used throughout all of Meteor.

While Meteor is a full-stack platform, it is modular by design and can have modules added, replaced, and removed. Meteor consists of the following primary subprojects:

- **Blaze:** Reactive UI library
- **Tracker:** Reactive programming
- **DDP:** WebSocket-based data protocol
- **Livequery:** Live database connectors
- **Full stack database drivers:** Use the same database API on client and server
- **Isobuild:** Unified build system for browser, server, and mobile
- **Accounts:** Instant account system for your app
- **Mini databases:** In-memory JS reimplementations of popular databases

These will be explored more fully in the relevant sections below.

If there were components of the workspace that were resource intensive these could be separated into microservices that run on different servers and communicate with the Meteor server over DDP. This means the main application would be responsive to users though it could introduce latency issues dependent on network speeds. Meteor also scales horizontally by being able to be run in a cluster.

Clients connect to the server by going to a URL in their browser. It loads as any normal webpage. Meteor packages all the needed HTML, CSS, and JavaScript within the webpage. A down side of this is that the package includes everything, even if it's not being used on that page of the app. It is possible to use a package to optimise this though it was not done for the prototype. These URLs can be shared through the usual means and there is also an index page on the server that lists the available workspaces and rooms. On this page as well is a form for creating a new room by specifying its name and the workspace it will be based on. See figure 3.2.

### 3.2.2 Templates

Meteor's default package, Blaze, for templates are effectively event-driven HTML pages for specific items—e.g. components and data sources. This gives a large amount of flexibility as is evidenced in the wide array of applications that have been made on the web based on HTML5 mixed with CSS and JavaScript.

The structure is defined using standard HTML elements and custom ones are possible using WebComponents. Along with these are reactive variable references interspersed throughout the templates which will be updated automatically if the reactive variable changes. These reactive variables are maintained by the Meteor server and follow the reactive programming paradigm. Instead of re-rendering the whole page only the affected element is re-rendered.

It's this reactivity and updating templates that enables collaborative work together as the effects of what everyone is doing is reflected across all user's devices, otherwise known as WYSIWIS and WYSIWID by Gutwin, Greeberg, and Roseman.

Styling is done through CSS. See code sample A.1 for an example component's structure. Each template has event handlers (for standard DOM events) and helpers defined in JavaScript. This separation of logic ensures that code can be easily written and maintained. See code sample A.2 for an example component's helpers and event handlers.

As mentioned, Meteor packages can be easily swapped out. An alternative to Blaze is ViewModel which uses a declarative and reactive event approach. The benefit of this approach is the tedious nature of event handlers and the verbose, imperative manner needed to create them.

With ViewModel, the state of the UI (template) is kept in a model and the UI elements are bound to properties of that model. This takes advantage of Meteor's inherent reactivity. For example, instead of writing an event handler for when a user clicks on a submit button and the form values are fetched using, the input fields themselves are bound to UI properties. ViewModel was tested and was found to be much preferable though there were some issues with dragging and for now standard

Blaze templates were used.

When a room is created, the server goes through the associated component schemas, finds their templates, and renders them on the canvas. These templates are then bound to instances of the schemas.

For distributing UI elements its quite simple in that the elements templates is rendered by itself, ‘outside’ of the canvas. With appropriate CSS for responsive design the element will respond to the type of device such as its size and input devices.

### 3.2.3 Schemas

Schemas are validated to make sure they have the required properties to ensure that everything within the concept is correct otherwise if invalid objects were not checked for the workspace would not function correctly. There are collections for workspaces, communication channels, rooms, and components. When the server start it inserts into these collections pre-made examples though it’s possible to insert more at any time though if a workspace is updated to have another component it will not be available in existing rooms.

### 3.2.4 Usable computers and devices

Any computer or device that has a browser that supports HTML5, CSS3, and JavaScript, and has a network connection can be used. Due to the focus on standards and compatibility this includes all major browser versions from 2009: Internet Explorer  $\geq 8$ , Firefox  $\geq 31$ , Chrome  $\geq 31$ , Safari  $\geq 7$ , Opera  $\geq 29$ , iOS Safari  $\geq 7.1$ , Opera Mini  $\geq 8$ , Android Browser  $\geq 4.1$ , and Chrome for Android  $\geq 42$ .

Input devices are primarily mouse and keyboard or touch screen. Basic touch screen support was tested on iPhones and Android and found to be compatible without needing JavaScript libraries specifically for touch screen interactions such as Hammer.js. [42]

### 3.2.5 Users

A user is someone who has loaded the webpage which starts a Meteor session. Depending on if the workspace requires authentication, they’ll need to be logged in and if not, will be redirected to the index page. Only standard participant roles will be implemented using Meteor’s default account system. Roles were tested though by using the `meteor-roles` package where a role is given a name and user’s IDs are assigned to that role. As needed, a function can be called to check if a user is in a role.

### 3.2.6 Data protocol

HTTP connections aren't stateful and not persistent. Once a request has finished—successfully or not—the connection between server and client is closed. WebSockets are a way to open a session between client and server that is not closed where messages can be sent/received and acted on—any type of message can be sent, there is no defined protocol. With HTTP there are defined actions (or, 'verbs') such as GET, POST, UPDATE, and DELETE. This allows for easy handling of the types of actions that occur between clients and servers.

The Distributed Data Protocol (DDP) that is used by Meteor can be thought of as 'REST for WebSockets'. [41] [43] This brings the benefit of both HTTP actions, and WebSocket connections together. Another benefit is that the server can track the state of the user which previously required using cookies and other methods. DDP is used for synchronising data (using the publish/subscribe model) and remote procedure calls on the server.

There are other protocols such as ZeroMQ, WebRTC, and WAMP which have different though sometimes overlapping purposes. DDP's main focus being data synchronisation suits the workspace example as it needs to have the data (components, states, etc.) synchronised throughout the clients. The WebRTC protocol focuses on multi-media streaming and could be useful audio and video communication channels. [44] Also, WAMP allows for the server to call procedures on the client instead of only the other way around. [45] This would have been useful for the example to be able to call the Arduino to run a function instead of having that function run every few seconds.

DDP is very simple, it's simply specific JSON messages, which means it can be easily implemented on anything that supports WebSockets. While there exists DDP client libraries in the most common programming languages [46] there weren't any yet for embedded devices which would be necessary for easy communication from data sources to physical objects. To demonstrate DDP on an embedded device and for the purpose of the example I implemented most of the DDP specification in C++ on an Arduino Mega. An Arduino was chosen due to the wide support and use—for example, it has a WebSocket library [47] and a JSON library [48] that were up building the implementation. The downsides of the Arduino Mega is that it has relatively slow processing speed and with the overhead of the libraries and time for string processing functions created a small latency. Also, storage is very minimal and an SD attachment was tried though this increased the delay slightly more as well. Depending on the workspace and the data source that the Arduino is connected though it may not be an issue if users are made aware that that specific data source has an inherent latency.

### 3.2.7 Real-time

Meteor’s real-time capability is possible by its combination of a reactive UI library (Blaze), reactive programming (Tracker), and pub/sub with DDP.

Tracker acts as a mediator between reactive data sources and reactive data consumers. [41] For example, using livequery and MongoDB any changes are reflected in templates. Meteor provides Tracker integrated with Blaze and elsewhere and using it for other purposes is as easy as wrapping an existing function that returns a variable in a Tracker `autorun` function.

Meteor has built-in latency compensation in that any changes to collections are written to the minimongo collection and to the server’s collection. This way the affects of changes on collections can be shown immediately client-side so that it appears to the user that there is no delay. The server’s collection is the final source of truth for collections and if there are any conflicts the client side will be updated to the values in the server’s collection. This can cause disrupting user experiences though with simple locking, low-latency, and user coordination it is minimal.

For data sources to work the framework needs to be able to set up real-time connections to outside sources. This can be done by connecting to Meteor using DDP over a WebSocket.

A Meteor server was set up that displayed tweets from Twitter that contained specified hashtags. These tweets were stored in a collection that was filled by having a separate Node.js program running which used a Twitter library to find the tweets and then sent them over a DDP connection to the Meteor server using a DDP client library. [49] The only slow performance were the API limitations from Twitter, otherwise using DDP over WebSockets was fast though this was a low-volume traffic test. This shows that any software platform with a DDP library can send and receive real-time data from the Meteor server.

Due to bandwidth and performance limitations—including on the minimongo implementation—it’s beneficial to reduce the use of both as much as possible. Also, some data is ephemeral in nature and does not need to be stored persistently. Collections require for waiting for MongoDB operations (which are blocking) to be completed. A way of avoiding this is by having collections that are not stored in the server’s MongoDB and instead have its data transmitted by the server to all clients which have their minimongo DBs with the collections being stored. For example, with dragging a component, their positions are kept in a collection labelled ‘transient’ which is not stored in the server’s database but only in-memory. When a dragging ‘started’ even is fired the transient collections are updated with the new positions while the component is being dragged. When the dragging ‘finished’ even has fired the component instances collection is updated with the new final position. The speed differences went from having a noticeable  $\sim 50$  ms delay when dragging

to an imperceptible delay.

### 3.2.8 Data storage

MongoDB is a modern NoSQL database that uses a document model as opposed to the common relational DB. Instead of tables there are collections which hold one or more documents. [50]

Everything being a schema maps clearly to the document data model of MongoDB and with both being represented in JSON there is no conversion needed—how schemas are in the application are the same as how they are in the database and you can even apply schemas directly to MongoDB collections. A relational DB isn't expressly needed and areas where there are relationships, such as a list of objects that a workspace uses, are easily queried using the MongoDB query language. Therefore the default database with Meteor, MongoDB, was used. MongoDB offers 'atomic writes on a per-document-level' which suits the reactive nature of workspaces where single parameters of documents are regularly updated.

MongoDB isn't without its problems though which could show if a workspace was to have extremely large numbers of users, or, complex workspaces. Kyle Kingsbury found that "Mongos consistency model is broken by design: not only can strictly consistent reads see stale versions of documents, but they can also return garbage data from writes that never should have occurred." [51]

MongoDB is able to run on small amounts of RAM which has allowed the minimongo DB created by Meteor to run efficiently on mobile devices. When testing chat there was no noticeable performance different when waiting on writes to the server's MongoDB as opposed to only having the messages pass-through the server.

minimongo is a client-side, in-memory JavaScript implementation of MongoDB. It mirrors the MongoDB on the server. Latency compensation works by interacting with this database at the same time as with the server-side database. Another benefit by having local copies of data on the user's device is that any queries against the database are almost instantaneous as they do not need to be fetch from the server itself.

Databases are often a performance bottleneck or require extra logic to transform the results of queries into something that is displayed. Livequery are live database connectors which return results when the query is made, and sets up a stream for the query that contains messages when any create, update, or delete actions have been performed. Meteor has livequery for the MongoDB it ships and this is used with reactive programming for templates and elsewhere within the concept. Livequery works well with querying against client-side collection storages as it keeps them in sync with the server.

### 3.2.9 Chat communication

A simple text chat was implemented using a collection where messages are displayed next to the user's name. Messages being in a collection allows for history which benefits users who join late or are disconnected.

## 3.3 Example workspace: PWM/Colour theory

The example workspace that was used in user testing is one that's used for students to learn about pulse width modulation (PWM) and colour theory (red / green / blue). A colour is generated and shown in the left of the workspace along with instructions of what they're to do. By using the PWM Input object students can work together to find three values that will cause the RGB LED to be the desired colour. To find out what colour it is there is an RGB colour sensor in front of the LED which sends the R, G, and B values to the server. In the workspace are the following components:

- **Objects:**

- Power: Represents a power source for the electrical circuit
- Ground: Represents a ground source for the electrical circuit
- PWM Input: Three values for R, G, and B on the LED
- PWM Output: Graphs the PWM values that the students have sent to the RGB LED

- **Data sources:**

- RGB LED: Receives the R, G, and B PWM values
- RGB Sensor (RGB Output): Detects the R, G, and B values

A standard participant role was used along with text chat on the right-hand side. Collaboration is required for the students to agree upon the R, G, and B values and to send them to the LED—see figure 3.4 for a screenshot of the canvas. For example, if initially the sensor detects too much green then they know they need to decide how much to lower it by, which will help them understand how colours mix. The history graph of the used PWM values helps the students see the progression of colours. To demonstrate distributability the PWM Input (see figure 3.5) and RGB Output components have 'divide' links which will cause the respective component to rendered by itself.




jesse ▾

## Index

Here you can find the available workspaces and the existing rooms for them. Feel free to create a new room if you would like to work by yourself or with others.

### Start a new room

Name:

Workspace:  

## Workspaces

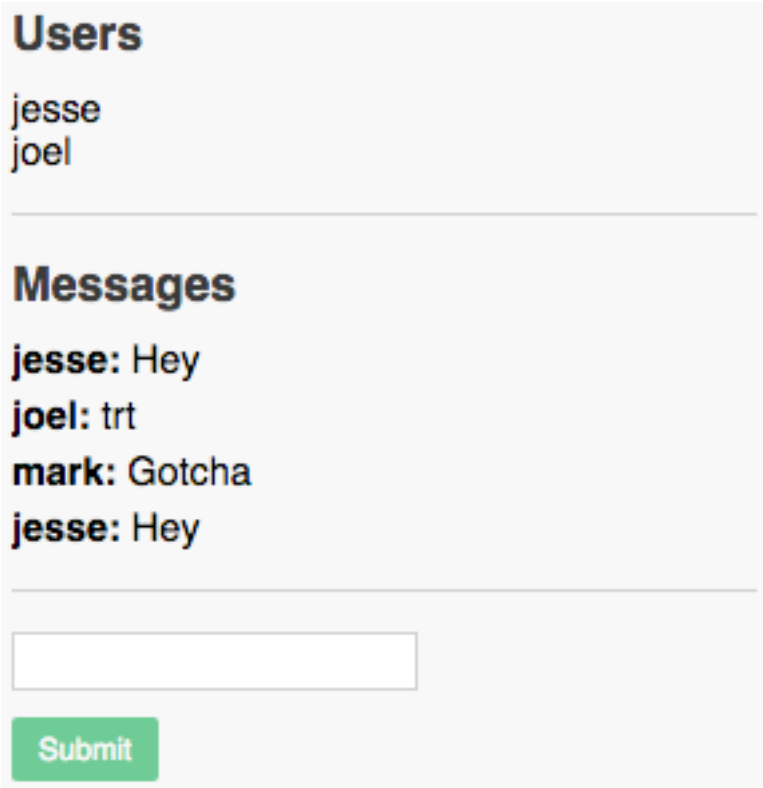
### PWM/Colour theory

Students learnt about PWM and colour theory by controlling an RGB LED.

#### Current rooms

<b>Demo</b> (1 / 5 current users)	<input type="button" value="Join"/>	<input type="button" value="End"/>
<b>Mark</b> (1 / 5 current users)	<input type="button" value="Join"/>	<input type="button" value="End"/>
<b>Test</b> (1 / 5 current users)	<input type="button" value="Join"/>	<input type="button" value="End"/>
<b>Em's group 1</b> (1 / 5 current users)	<input type="button" value="Join"/>	<input type="button" value="End"/>

Figure 3.2: Server index of available workspaces and rooms



The image shows a chat room interface with a light gray background. It is divided into three horizontal sections by thin gray lines. The top section is titled "Users" in bold black text and lists two names: "jesse" and "joel". The middle section is titled "Messages" in bold black text and lists four messages: "jesse: Hey", "joel: trt", "mark: Gotcha", and "jesse: Hey". The bottom section contains a white text input field and a green "Submit" button.

**Users**

jesse  
joel

---

**Messages**

jesse: Hey  
joel: trt  
mark: Gotcha  
jesse: Hey

---

**Submit**

Figure 3.3: Chat room with users and messages

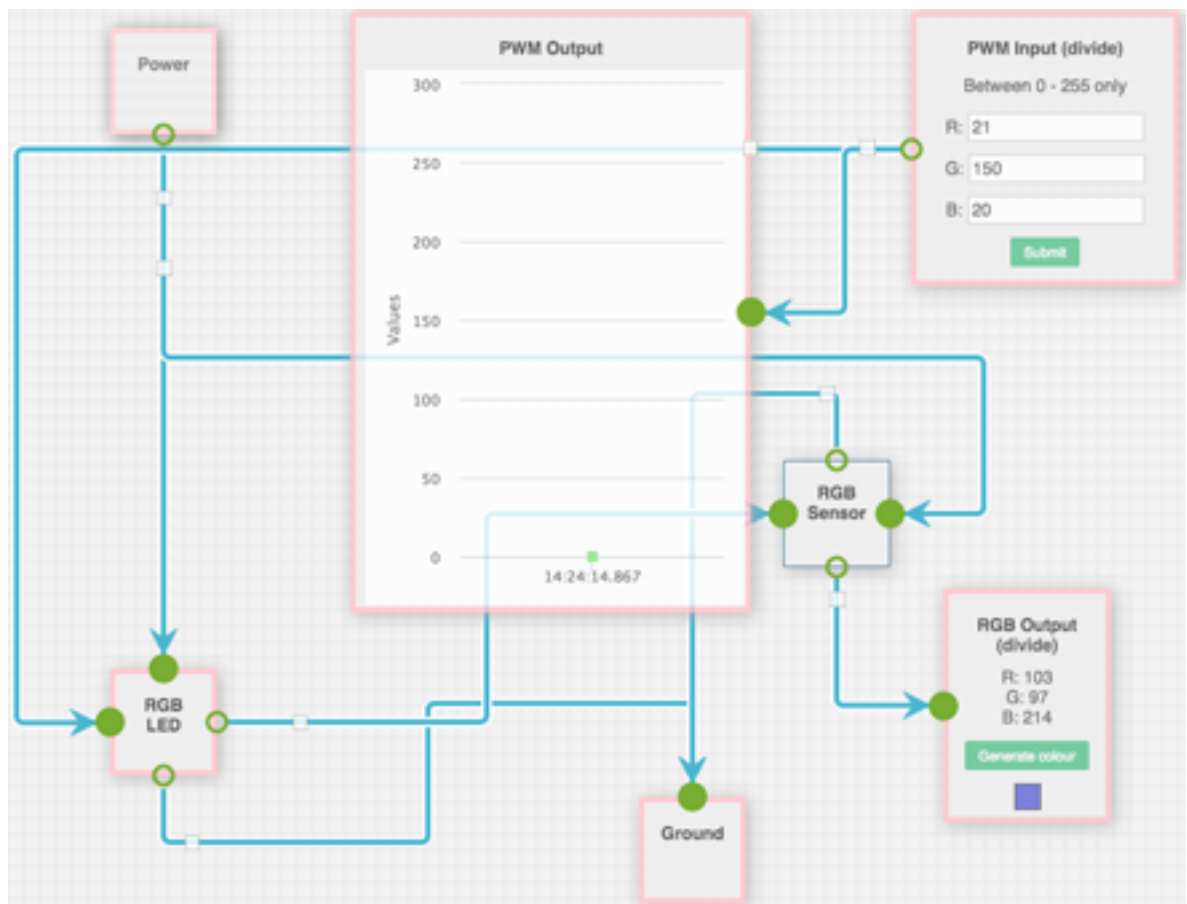
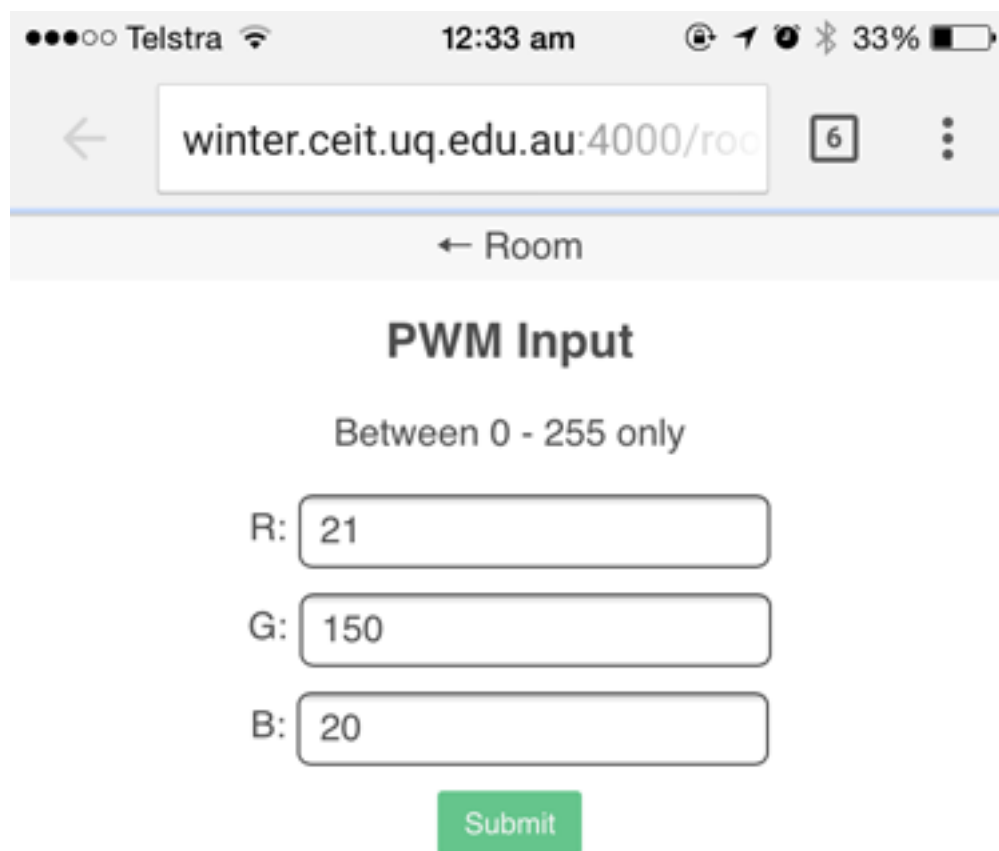


Figure 3.4: Example canvas



The image shows a screenshot of an iPhone screen displaying a web browser. The status bar at the top shows the carrier as Telstra, the time as 12:33 am, and the battery level at 33%. The browser's address bar contains the URL `winter.ceit.uq.edu.au:4000/roo`. Below the address bar, there is a back arrow and the text "Room". The main content area of the browser displays a form titled "PWM Input". Below the title, it says "Between 0 - 255 only". There are three input fields labeled "R:", "G:", and "B:". The "R:" field contains the value "21", the "G:" field contains "150", and the "B:" field contains "20". Below these fields is a green "Submit" button.

**PWM Input**

Between 0 - 255 only

R:

G:

B:

Figure 3.5: DUI example: single component on an iPhone

# Chapter 4

## User testing and results

### 4.1 User testing

People from various areas were given time to use the prototype with the ‘Thinking aloud’ usability testing method used to evaluate how well the user grasped the concepts. [52] Afterwards they were also interviewed to evaluate the validity and usefulness of the thesis from an applied point of view.

#### 4.1.1 Distance education use

Julia Fuglsang, an arts teacher at the Brisbane School of Distance Education, was interviewed at her workplace to gain an understanding of the current state of software used for distance education in Australia.

Distance education primarily relies on the government software, iConnect. Its primary functionality includes, but is not limited to:

- Screen sharing: All connected students can view the teacher’s screen. The teacher can grant control to one student at a time.
- File management: Includes student submission.
- Slideshow presentations.
- Whiteboard: Individual per user. Teacher’s can see student’s whiteboards.
- Communication: webcams and chat.

The teacher made the following critiques about iConnect:

1. Communication isn’t reliable due to the differing qualities of webcams and internet connections. Distance education students are usually in remote locations with poor internet quality.

2. No student-to-student interaction aside from chat.
3. Little teacher-student interaction. Teaching follows the “watch me do and repeat yourself” style.
4. Slow to be updated—e.g. does not work on tablets such as iPads.
5. Poor UI.
6. Whiteboard is only suited to simple work. Even text-related work is difficult—at the moment, students take screenshots of their Word documents and shows that. Aside from being a laborious process, the teacher isn’t able to edit the document.
7. Hard to integrate iConnect with external media such YouTube videos.

The teacher used the prototype and in general was impressed as it helped with most of the critiques above. Specifically, with critique 3, moving beyond screen sharing and allowing the teacher and students to work collectively would enable her to more effectively guide students directly/hands-on as opposed to them having to hear her verbal instructions and attempt the activities by themselves. The teacher was invited to use her own mobile device to join in on the prototype and said that students not requiring to buy new hardware, and it also not being dependent on specific hardware-type, would help with students being at similar learning levels. Related to this, one roadblock she finds is that when teaching concepts that relies on having some real-world item present, for example, paint for colour theory, she would often find that not all students had the same paints available causing her to have to repeat different parts to different students. Having a common workspace such as a standard colour wheel object and paint brush in the workspace would allow her to teach the concept once and then the students can apply the theory specifically to their paints with her answering questions and guiding along. The teacher’s current software environment uses text and audio/video communication channels though they’re not integrated into the UI which makes it hard for the teacher to manage communication. She said that having the chat functionality as part of the main UI helped with this.

Students being able to have their own workspace to use in their own time, and then invite the teacher to join it to observe, would help with critique 1. Also, with any type of object being able to be defined and created using HTML/CSS/-JavaScript, different work types can be handled from basic text processing (e.g. Word) to more advanced types. With web browsers now being fist-class apps on computers and mobile devices and having high performance, critique 4 would be

alleviated as all computers and devices generally have the same browser-based capabilities. Using a browser allows for easy importing and manipulation of external media as desired from critique 7. The relatively small bandwidth needed for websites compared to video benefits those students who live in remote areas with limited internet connections.

Her main suggestion was that a higher-fidelity prototype would have helped her known how to interact with the different parts of the concept—e.g. objects that looked similar to their real-world part) would've helped her know how to interact with the different components.

### 4.1.2 In-class use

A primary school teacher from Education Queensland, Emily Rogers, was shown the prototype gave overall positive feedback and detailed different in-class difficulties that would be alleviated. While Education Queensland is preparing interactive PDFs and videos to assist with teaching the new state curriculum, they're limited in the effectiveness in how they can be taught in class, and with the different learning styles of children. If the curriculum had preset workspaces that could be used it would have the desired consistency amongst classes and schools while giving teachers flexibility in teaching and allowing students to learn in a more hands on way. An 'explicit I/We/You do teaching' method is used where teachers first explain concepts ('I'), then the students and teacher work together ('We'), and finally students work by themselves or in groups ('You') which is well suited to the workspace concept.

When students are working in groups there's often small behavioural conflicts such as some students not being focused, simply playing around with the lessons, or interfering with other students things. A divisible UI would help here as she would be able to separate parts of the lessons to the students own device. Some students disengage as well when they feel they aren't doing as well as others and aren't as smart as them. Due to the open nature of desks and the classroom how much progression a student is making is very visible. Students having their own rooms (workspace interfaces) on their devices, that the teacher can observe and participate in, allows those students to feel more comfortable and to learn at their own pace.

Digital whiteboards and other technology often takes a considerable amount of time to be set up, especially when used frequently for different curriculum topics. With workspaces being quickly created from defined schemas, it would help teachers in their class preparation. Emily suggested that workspaces could take parameters/content. For example, if there was a workspace for learning English, it would be set up for doing so and the teacher could simply drop in the current book or so that they would be using.

From experience she has found that students, especially young boys, learn better when they can see real world applications of the principles. Having simulations in the workspace, especially when connected to real world objects, would help them learn.

All students progress and marks are tracked through Education Queensland's program OneSchool and often requires a lot of time to keep up-to-date. If progress could be updated by tracking what students do in the workspaces, or have assessment done through workspaces (even with automatic marking depending on the lesson, such as spelling), would save teachers much time. This brings about an interesting topic that I had not considered—analytics within workspaces such as tracking user's inputs and object manipulations.

For in-class use the communication channels are not necessary as there is face-to-face communication.

### 4.1.3 Out-of-class use

Students often rely on Google Docs to work on assessment together or visit laboratories to practice concepts. Stephanie Greer, a student at The University of Queensland, was shown the prototype and said that it would be helpful to be able to collaboratively work on concepts that were previously impossible aside from being on campus would be very helpful, especially so when working together with other students. She could easily use the prototype though the subject (PWM and colour theory) was new to her.

## 4.2 Results

From the user testing and functionality of the prototype it can be seen that the use of the Meteor application framework in a client-server architecture greatly suits this type of problem.

One possible issue for adoption would be internet speeds though globally in 2014 the average web speed was 4.5 Mbps which is more than sufficient. [53] Otherwise, there is also the issue of language compatibility and accessibility for disabled persons though this would be more dependent upon the designers as the technology exists to support these issues.

While performance was not a critical issue for the prototype there was the previously mentioned delay when dragging objects which was optimised. Meteor has been tested 200 [54] and 20,000 [55] concurrent users and performed smoothly.



# Chapter 5

## Conclusion and future work

The prototype has been shown to enable collaborative work by using real-time synchronisation of UI elements seamlessly and efficiently in a way that can be expanded to many different application and interaction uses. The use of communication channels assists with collaboration, ensuring working with mutual flexibility and conflict resolution.

The concept of the workspace gives great flexibility in application and in user testing they were excited for the many different ways in which they could see themselves using it. This is mostly due to the overall framework and the focus on everything being a schema and the loose-coupling from representation (schemas) to implementation

WebSockets with DDP has shown to be a reliable, efficient, and easy way of synchronising data. Reactive programming is suited well to real-time applications as it tightly binds the data and UI together as opposed to event-driven programming which treats things in a more time-delayed manner.

While it was previously difficult and required specialised implementations to have a distributed user interface it's been shown that using a web server and web pages, with responsive design, this can be achieved in a simple fashion. The use of web pages as well enables almost any modern device to be used, not requiring programming and graphic libraries that otherwise might not be available or have high performance costs.

Distributing UI elements was a seamless experience from within the workspace and the distributed state as a URL is an easy way to access the elements from any device.

I was able to meet the goals for the proposed concept. For possible future work, as the distance education teacher mentioned, the design would be key to adoption. As design is “.. not just what it looks like and feels like. Design is how it works.”, [56] a well thought-out and implemented design would be the crossover point between the functionality being effective as far as it is actually utilised by users. [57]

Design is also responsible for enabling accessibility for all user types. For example, it would be wise to investigate how well languages aside from English work and how those with motor skill difficulties would be able to manipulate the objects and data sources in an efficacious way and with being able to ‘keep up’ in a real-time environment.

OT has been proven to work with text and similar data types [58] and it would be worthwhile to investigate if it can be expanded to user interactions which is more complex due to the cause-effect of events from interactions. Similarly, instead of a client-server architecture for real-time, a peer-to-peer architecture could be considered. For having effective collaboration, experiments could be made in showing approval/disapproval on actions, like ‘Liking’ doing things, using emojis, and so on, for purpose of allowing the functions like smiling for eliciting confirmation. Implement showing other user’s cursors would increase the WYSIWIS/WYSIWID models.

While schemas were used extensively in the initial creation, instances of schemas could be serialised to store state information such as values and position which could be used to pause/resume workspaces, for backup, offline use, etc.

For the collaborative model it would be worth while to look into more intricate user models, and how this affects working together, etc. For example, with education, Mason, Rossman, Coppola et al. [22] identified three different tutor roles: content-related (provides information, explains concepts, ..), social (facilitate/maintain interest and motivation of the participants), and organisational support (planning/designing activities). These roles would need to have extra or different functionality which could alter how the users interact in real-time. Also, collaboration could be expanded from being users only interacting with other users in the room they’re in to having inter-connected rooms. The rooms could be instances of different workspaces and it would be interesting to mimic the flow often found in businesses and elsewhere where you have multiple workspace/user combination types whose results flow onto the next workspace/user combination.



# Appendix A

## Prototype

The full code for the prototype and example can be found in the following git repositories.

- <https://github.com/jesse-c/thesis-workspace>
- <https://github.com/jesse-c/thesis-sensor>
- <https://github.com/jesse-c/Arduino-DDP>

Selected code samples.

### Code samples A.1: PWM input component

```
<template name="object__pwm_input">
  <div class="{{class}}" id="{{id}}" data-id="{{_id}}"
    style="top:{{top}}px;left:{{left}}px">
    <h1>{{name}} <a href="{{objectLink__id}}"
      target="_blank">(divide)</a></h1>
    <p>Between 0 - 255 only</p>
    <form>
      <div class="input__field">
        <label>R:</label>
        <input type="text" id="input-r" value="{{r}}"
          disabled="{{inUse}}">
      </div>
      <div class="input__field">
        <label>G:</label>
        <input type="text" id="input-g" value="{{g}}">
      </div>
      <div class="input__field">
        <label>B:</label>
        <input type="text" id="input-b" value="{{b}}">
      </div>
    </form>
  </div>
```

```

    </div>
    <input type="submit" id="pwm_input__submit"
          class="action_primary">
  </form>
</div>
</template>

```

#### Code samples A.2: PWM input component

```

Template['object__pwm_input'].helpers({
  r: function() {
    return R.findOne().value;
  },
  g: function() {
    return G.findOne().value;
  },
  b: function() {
    return B.findOne().value;
  },
  objectLink: function(id) {
    var room = Rooms.findOne();
    return '/room/' + room._id + '/object/' + id;
  }
});

Template['object__pwm_input'].events({
  'focus_#input-r': function() {
  },
  'change_#input-r': function() {
    R.update(R.findOne()._id, { $set: {
      value: $('#input-r').val() } });
  },
  'change_#input-g': function() {
    G.update(G.findOne()._id, { $set: {
      value: $('#input-g').val() } });
  },
  'change_#input-b': function() {
    B.update(B.findOne()._id, { $set: {
      value: $('#input-b').val() } });
  },
  'click_#pwm_input__submit, _submit_form': function(e) {
    e.preventDefault();
  }
});

```

```
var newR = $('#input-r').val();
var newG = $('#input-g').val();
var newB = $('#input-b').val();

R.update(R.findOne()._id, { $set: {
  value: newR } });
G.update(G.findOne()._id, { $set: {
  value: newG } });
B.update(B.findOne()._id, { $set: {
  value: newB } });

Submitted.insert({
  r: newR,
  g: newG,
  b: newB,
  shown: false
});

Meteor.call('updateStatus', 'Submitted_PWM_values');
}
});
```

# Appendix B

## Example

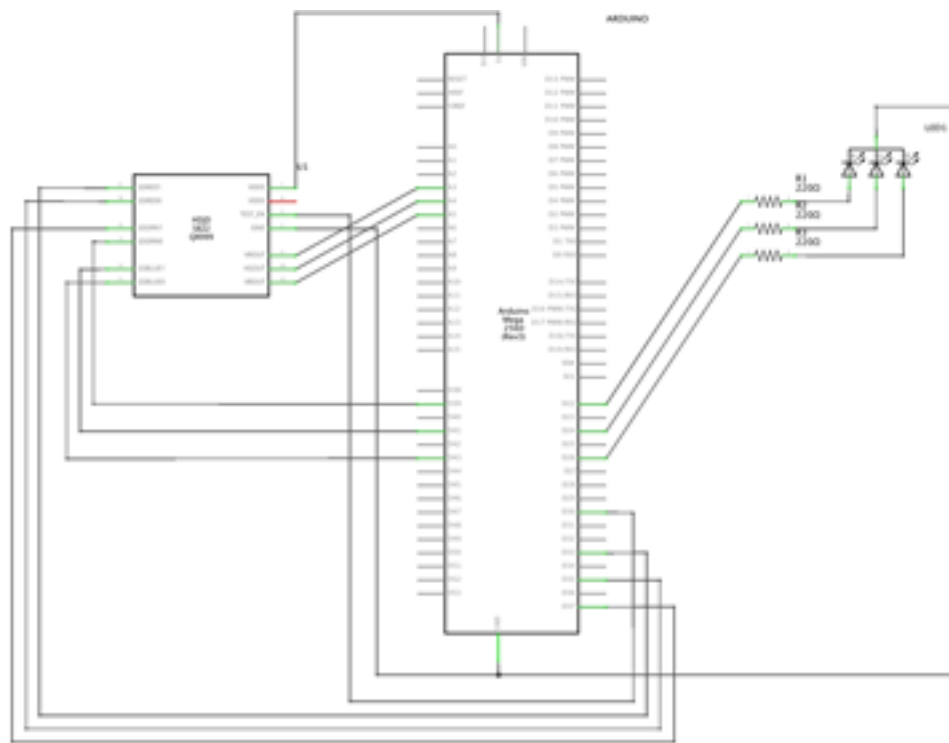


Figure B.1: Electronics schematic





# Bibliography

- [1] “Microsoft office online - word, excel, and powerpoint on the web.” <https://office.live.com/start/default.aspx>. (Visited on 06/15/2015).
- [2] “icloud.” <https://www.icloud.com/>. (Visited on 06/15/2015).
- [3] S. L. O. N. D. P. S. J. James Manyika, Jacques Bughin and S. Ramaswamy, “Global flows in a digital age,” tech. rep., McKinsey Global Institute, April 2014.
- [4] X. Qiu and A. Jooloor, “Web service architecture for e-learning,” *Journal of Systemics, Cybernetics and . . .*, vol. 3, no. 5, pp. 92–101, 2006.
- [5] R. Tesoriero and M. Lozano, *Distributed User Interfaces: Collaboration and Usability*. 2012.
- [6] M. Weiser, “The computer for the 21st century,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, 1999.
- [7] F. Chimero, “The webs grain.” <http://www.frankchimero.com/writing/the-webs-grain/>, 2015. (Visited on 06/10/2015).
- [8] “Precursorfast prototyping web app, makes collaboration easy..” <https://precursorapp.com/>. (Visited on 06/10/2015).
- [9] “Apache wave.” [https://en.wikipedia.org/wiki/Apache\\_Wave](https://en.wikipedia.org/wiki/Apache_Wave). (Visited on 06/10/2015).
- [10] “Etherpad.” <https://en.wikipedia.org/wiki/Etherpad>. Accessed: 2014-08-26.
- [11] C. Sun and C. Ellis, “Operational transformation in real-time group editors: issues, algorithms, and achievements,” *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pp. 437—446, 2004.
- [12] “Meteor.” <https://www.meteor.com/>. Accessed: 2014-08-28.

- [13] “Derbyjs.” <http://www.derbyjs.com/>. Accessed: 2014-08-24.
- [14] “Sharejs.” <http://www.sharejs.org/>. Accessed: 2014-08-20.
- [15] “Firebase.” <https://www.firebase.com/>. Accessed: 2014-08-02.
- [16] “Simperium.” <https://www.simperium.com/>. Accessed: 2014-08-24.
- [17] “Cscwmatrix - computer-supported cooperative work - wikipedia, the free encyclopedia.” [https://en.wikipedia.org/wiki/Computer-supported\\_cooperative\\_work#/media/File:Cscwmatrix.jpg](https://en.wikipedia.org/wiki/Computer-supported_cooperative_work#/media/File:Cscwmatrix.jpg). (Visited on 06/15/2015).
- [18] R. Baecker, *Readings in Human-computer Interaction: Toward the Year 2000*. Interactive Technologies Series, Morgan Kaufmann Publishers, 1995.
- [19] C. Gutwin, S. Greenberg, and M. Roseman, “Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation,” *People and Computers*, pp. 281–298, 1996.
- [20] J. Allwood, “Cooperation and Flexibility in Multimodal Communication,” in *Multimodal Cooperative Communication*, vol. 2155, pp. 113–124, 2001.
- [21] C. Bouras, P. Lampsas, P. Zarafidis, and A. Zoura, “Web-Enabled Distributed Collaborative Learning Environment,” *Proceedings of ICCE’98 (1998 ...*, pp. 1–4, 1998.
- [22] M. Gerber, S. Grund, and G. Grote, “Distributed collaboration activities in a blended learning scenario and the effects on learning performance,” *Journal of Computer Assisted Learning*, vol. 24, no. 3, pp. 232–244, 2008.
- [23] G. Johnson, “Instructionism and Constructivism: Reconciling Two Very Good Ideas,” *Online Submission*, 2005.
- [24] J. Bransford, A. Brown, and R. Cocking, “How people learn: Mind, brain, experience and school, expanded edition,” *DC: National Academy Press, Washington*, 2000.
- [25] S. Papert and I. Harel, “Situating constructionism,” *Constructionism*, 1991.
- [26] “ilabs at uq — ceit.uq.edu.au.” <http://ceit.uq.edu.au/content/ilabs-uq>. (Visited on 06/11/2015).
- [27] J. Ma and J. V. Nickerson, “Hands-on, simulated, and remote laboratories,” *ACM Computing Surveys*, vol. 38, no. 3, pp. 7–es, 2006.

- [28] R. Rädle, H.-c. Jetter, and H. Reiterer, “TwisterSearch : A distributed user interface for collaborative Web search,” pp. 1–17.
- [29] “Supreme commander (video game) - wikipedia, the free encyclopedia.” [https://en.wikipedia.org/wiki/Supreme\\_Commander\\_%28video\\_game%29](https://en.wikipedia.org/wiki/Supreme_Commander_%28video_game%29). (Visited on 06/11/2015).
- [30] “Xbox smartglass — turn mobile devices into second xbox screens.” <http://www.xbox.com/en-US/smartglass>. (Visited on 06/11/2015).
- [31] “Wii u - wii u game pad - nintendo.com.au.” <http://www.nintendo.com.au/wii-u-hardware-features-game-pad>. (Visited on 06/11/2015).
- [32] “Apple - osx yosemite - mac + ios continuity.” <https://www.apple.com/au/osx/continuity/>. (Visited on 06/11/2015).
- [33] “Mobile technology fact sheet — pew research center.” <http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/>. (Visited on 06/11/2015).
- [34] E. Marcotte, “Responsive web design an a list apart article.” <http://alistapart.com/article/responsive-web-design>. (Visited on 06/11/2015).
- [35] V. Tran, J. Vanderdonckt, M. Kolp, and S. Faulkner, “Generating User Interface from Task, User and Domain Models,” *2009 Second International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies, and Services*, pp. 19–26, 2009.
- [36] A. Peñalver, E. Lazcorreta, and J. López, “Schema driven distributed user interface generation,” *Proceedings of the 13th ...*, 2012.
- [37] “Welcome to microsoft pixelsense.” <https://www.microsoft.com/en-us/pixelsense/default.aspx>. (Visited on 06/11/2015).
- [38] H. M. Fardoun and A. P. Ciprés, “CSchool : DUI for Educational System using Clouds,” pp. 35–38, 2012.
- [39] J. P. Morrison, “Flow-Based Programming,” *Journal of Application Developers News*.
- [40] “The introduction to reactive programming you’ve been missing.” <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>. (Visited on 06/13/2015).
- [41] “Meteor.” <https://www.meteor.com/>. (Visited on 06/13/2015).

- [42] “Hammer.js - hammer.js.” <https://hammerjs.github.io/>. (Visited on 06/13/2015).
- [43] “Ddp specification version 1.” <https://github.com/meteor/meteor/blob/devel/packages/ddp/DDP.md>. (Visited on 06/13/2015).
- [44] “WebRTC.” <http://www.webrtc.org/>. (Visited on 06/13/2015).
- [45] “Wamp - web application messaging protocol.” <http://wamp.ws/>. (Visited on 06/13/2015).
- [46] “meteor/ddp.md at devel meteor/meteor.” <https://github.com/meteor/meteor/blob/devel/packages/ddp/DDP.md>. (Visited on 06/13/2015).
- [47] “brandenhall/arduino-websocket.” <https://github.com/brandenhall/Arduino-Websocket>. (Visited on 06/13/2015).
- [48] “bblanchon/arduinojson.” <https://github.com/bblanchon/ArduinoJson>. (Visited on 06/13/2015).
- [49] “oortcloud/node-ddp-client.” <https://github.com/oortcloud/node-ddp-client>. (Visited on 06/13/2015).
- [50] “Faq: Mongodb fundamentals mongodb manual 3.0.4-rc0.” <http://docs.mongodb.org/manual/faq/fundamentals/>. (Visited on 06/13/2015).
- [51] “Call me maybe: Mongodb stale reads.” <https://aphyr.com/posts/322-call-me-maybe-mongodb-stale-reads>. (Visited on 06/13/2015).
- [52] J. Nielsen, “Thinking aloud: The #1 usability tool.” <http://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/>. (Visited on 06/09/2015).
- [53] “The akamai state of the internet report.” <http://www.akamai.com/dl/soti/q4-2014-executive-summary.pdf>, 2015. (Visited on 06/13/2015).
- [54] “Meteor scalability - google groups.” <https://groups.google.com/forum/#!msg/meteor-talk/FcvWdfMRrvM/LeR7PHW0mRIJ>. (Visited on 06/14/2015).
- [55] “Scaling meteor to 20,000+ users in 7 days.” <http://blog.differential.com/scaling-meteor-to-20000-users-in-7-days/>. (Visited on 06/14/2015).
- [56] R. Walker, “The guts of a new machine - nytimes.com.” <http://www.nytimes.com/2003/11/30/magazine/the-guts-of-a-new-machine.html>. (Visited on 06/09/2015).

- [57] J. Nielsen, *Designing Web Usability: The Practice of Simplicity*. Thousand Oaks, CA, USA: New Riders Publishing, 1999.
- [58] “ottypes/docs.” <https://github.com/ottypes/docs>. (Visited on 06/15/2015).