Accessible Pointing Approaches for Web Applications

Alexander Birch

November 2013

Contents

1 Problem

1.1 Problem Context

Some domain knowledge will be divulged in this section, prior to describing the problem under investigation. Namely, the case for accessible navigation alternatives' necessity, and also the relevance of the web in contributing to the size of the problem.

Need for accessibility Computer users are not uniform in their capabilities and preferences. For this reason, users differ in the input mechanisms they choose for interacting with software. Some users may prefer to perform tasks using a keyboard rather than using a mouse, for example. Commonly mouse and keyboard will be recruited together. However some users' input strategies are constrained by factors such as disability: occupational syndromic conditions such as Repetitive Strain Injury (RSI) can preclude the ability to use a mouse, keyboard or both [1]. RSI and related disabilities are growing in prevalence, representing by some estimates 22% of people [2]. This disability accounts for one third of workers' compensation costs in the US private industry [3].

When the user cannot or will not use the primary input scheme of the software, it becomes necessary to provide alternative input modes. For simple interfaces that use standard components, keyboard and mouse can be both supported implicitly without additional development effort. Conversely, complex interfaces or ones involving non-standard components require explicit treatment from the developer to ensure that all funcitonality can be accessed. In cases where accessibility is not a priority or concern of the software developer, users can be left unable to use said software.

Rise of web applications Even as early as 2001, software made the leap to the web [4]. Services such as email, banking and word processing are being produced as web applications, due to a few advantages:

- high cross-platform, cross-device support & penetration
- centralized maintenance and upgrade
- centralized storage of user data
- (consensual) monetization of user information, telemetry
- scalability (down and up)
- zero end-user installation
- redundancy

Email clients such as Outlook [5] benefit from this; emails can be accessed from any web-enabled device without any need for installation, all users can be kept on the latest version without effort on their part, and user data can be analyzed to improve content filters or provide relevant advertisement. At peak times of operation, more servers can be recruited, and vice-versa, such that the cost of operations can be finely controlled based on usage.

1.2 Problem Description

This transition away from native applications is leading to sacrifices in accessibility. Web applications run within a native application (the browser), so there is conflict when it comes to the following accessibility paradigms:

- menu actions
- keyboard shortcuts
- tab order
- citizenship amongst other applications

Even on the conceptual level, there are problems with using the web to serve applications: a web application tries to fit a full application interface inside a web browser's (already saturated) interface. It is a second-class citizen; a user cannot 'switch application' to it, only trawl for it amongst the tabs and windows of some web browser.

Additionally, web applications are often guilty of foregoing standard UI components in favour of a bespoke UI made in JavaScript or CSS. This can be used to deliver impressive effects like transitions, or serve complicated shapes of interactive content or buttons. But this freedom makes it all too easy to produce inaccessible elements, such as buttons that can't be pressed or focused using the keyboard.

1.3 Existing Approaches

There exist at present a few categories of solution to the problem of web accessibility:

- following web development guidelines to produce a standards-compliant website
- using web technologies to create a non-standard interaction behaviour
- making the web browser behave in a non-standard manner

1.3.1 Recommended web practice

Guidance is provided on how to develop accessible web applications, in the form of the World Wide Web Consortium (W3C)'s Web Content Accessibility Guidelines (WCAG) [6] [7]. A description and analysis on its recommendations for keyboard access follows:

Pointing with access keys Access keys are one paradigm used for keyboard accessibility in the HTML5 specification [8]. Hotkeys can be assigned for the focusing of interface elements, used in conjuction with modifier keys. This system allows the user to jump instantly between distant or unrelated interface elements. However, its usefulness is predicated on the discoverability and availability of said key bindings.

It is hard to avoid conflicts with existing shortcuts used by the user's accessibility technologies, since there is no standardisation between browsers as to which modifier keys to use [9]. Recommendation exists that access keys be avoided altogether for this reason. Another problem with access keys is that they rely on explicit implementation by developers. They also require learning of a layout, and although attempts have been made at increasing standardisation [10], that standardisation is not prevalent.

Making interface tab-accessible It is difficult to develop a tab-accessible web application [11]. Since the only pointing expression assumed is the tab key, layout must be linearized. Not only is this hard to design and develop for (requiring explicit effort and markup changes), the end result is not that impressive; content that is far away in the tab order unavoidably requires repeated hammering of the tab key (making it far harder to reach with the keyboard than with a pointing device). Plus linearization doesn't suit web content, which often has navigation bars, side bars, feeds or some similar multi-dimensional layout.

Short-circuiting tab journey with skip-links An exception to the content linearization problem is skip links, which can short-circuit the tab journey around a page by allowing the user to choose alternative insertion points. These again require effort and understanding from the developer. It is hard to provide a suitable set of skip links, and again the best case is still a hamfisted pointing experience, where the user can't predict where they are going to or from.

Conclusion Even a website designed with accessibility standards in mind can be quite hard to navigate; web standards prescribe only a very limited amount of keyboard expression, so meeting these provides a very limited experience.

1.3.2 Using web technologies to augment accessibility

Whilst it is possible (to an extent) to create key bindings in web content (through access keys section 1.3.1, or through JavaScript event libraries like MouseTrap [12]), for the most part web applications do not make use of this. Possible reasons for this are:

- no agreed standard exists for how websites should provide keyboard shortcuts
- web application shortcuts are likely to conflict with native application's shortcuts
- existing native application shortcuts are hard to tiptoe around, since any browser or extensions could be used
- some devices (for example smartphones) do not have the same expressive keyboard capability
- would only be available on those websites that chose to implement it
- websites not guaranteed to implement in a uniform way

1.3.3 Changing the web browser

Adding accessibility on top of web content can be achieved through a web browser extension. This can change the way content is treated so as to work better for the preferred input mode. In some implementations, the browser is extended through the use of web technologies, which can coexist with the content of the website. One such example is Type-Ahead-Find [13], which allows users to move keyboard focus to hyperlinks by typing the content of said hyperlink. Another example is Stylish [14], which allows users to redesign a website to suit their needs.

Web browser extensions solve some of the problems associated with using web technologies alone to augment accessibility: key bindings are uniform across the user's browsing experience, so a user can configure non-conflicting shortcuts. The solution is also easy to distribute via web browser extension repositories.

2 Proposed Solution

2.1 Approach

Half the problem is the limited amount of expression afforded to the keyboard; if tabbing is the only navigation method offered, then there is a severe constraint on what can be achieved. A means for utilising more keys is necessary. Unfortunately websites cannot provide this, as they cannot predict

which key bindings will be free on the user's computer. Nor should they try: each website would have to be learned individually.

The burden of reserving key bindings should fall on the web browser, or an extension thereof. Here it can be controlled and configured, as well as uninstalled should it cause conflicts. A novel shortcut scheme would be used to avoid conflicts, such as sequences rather than chords, or invoking the extension explictly as a precursor to action.

The rest of the problem is that accessibility is presently found only on websites that design for it. It is easy to blame developers for making websites that only work with a mouse or touchscreen, but the real problem here isn't that the website is designed wrong, it is that the keyboard isn't powerful enough to cope with these pointing situations.

Again, a solution where the keyboard is augmented via the browser is desirable. Increasing keyboard control diminishes the number of problem situations that exist for it.

2.1.1 What is novel about the approach?

Previous attempts (such as suggested best practice, described in section 1.3.1) at introducing keyboard accessibility to web applications have been focused on redesigning the website. The novelty of our approach is that we redesign instead the manner of traversing the website. Pointing will be designed from the ground-up to expect and support complex non-linear layouts using non-standard components.

2.1.2 Why does it make sense?

The approach embraces the way the world is making web applications (complex, bespoke, mouse-optimized interfaces), and attacks the problem that necessitated non-standard interfaces: that web applications are second-class citizens, described as 'content' in a pane of a native application. The pointing system will exist inside the context of the web application rather than the native application, and will acknowledge the web interface as the highest-level citizen. This makes for a clearer paradigm.

3 Methodology

3.1 Deliverables

The goal of this work is to produce an accessibility layer to sit between a web browser and web content, which would augment the accessibility of said content. The primary purpose of the accessibility layer is to achieve effective pointing via a sequence of keystrokes.

The system will aim to solve better the pointing situations to which tab navigation [1] is less suited. Pointing contexts such as form navigation, at which tab navigation is satisfactorily effective, need not be a focus of the novel system. Ideally the novel system would co-exist with tab navigation, allowing the user to fall back on tab navigation in cases where it is suitably effective (or more effective).

3.1.1 Software solution

The choice of browser for implementation is immaterial; the same logic can be applied to any extensible browser. This work will concern itself with Google Chrome, but the choice is arbitrary. The accessibility layer will be implemented in the form of a web browser extension, as this meets the criteria of sitting between browser and content, persists between page navigations, and doesn't incur work on the website developer's end.

3.1.2 High-level Requirements

This section defines the requirements of the software solution.

Priorities are as follows:

1. High- primary goal of work
2. Med- secondary goal of work
3. Low- goal not essential for work, but appealing nonetheless

Non-functional requirements

- **High** Software is easy for users to obtain and install
- **Med** Software features are easy to invoke
- **Low** Software is a good citizen (conflicting key bindings should be reconfigurable or non-existent)
- **Low** Default settings of software should cause minimal conflict in key bindings versus web browser
- **Low** Default settings of software should cause minimal conflict in key bindings versus accessibility technologies
- **High** Software should be resilient to bad markup
- **High** Software should work on websites that are not designed for accessibility

[1] Side-note: the term 'tab navigation' is used in this document always to refer to focusing interface elements via the tab key (sometimes known as 'tabbing navigation') — not to be confused with 'tab navigation', where a user switches browsing contexts represented as 'tabs' in their windowing system.

Functional requirements

- **High** Improve lookup time (compared to tabbing) of any given interface element
- **High** Improve lookup time (compared to tabbing) of interface elements far away from currently focused element
- **Med** Detect related interface elements, such as repeating elements in a list
- **Med** Provide a means to travel between related elements
- **Low** Improve lookup time (compared to tabbing) of related interface elements
- **Low** Improve lookup time (compared to tabbing) of unrelated interface elements

It must be noted that this pointing system is being designed for a different problem domain than tabbing; tabbing behaviour seeks the next focusable element found in a depth-first trawl of the webpage markup (excepting 'tabindex' overrides, which can be used to rewrite the order or move elements in and out of the tab order). This aspect of tab behaviour makes it very well-suited to traversing forms, where ideally the next element for interaction will be the next one in the markup. Problematically, the websites mix paradigms and use tabbing for pointing, which it is less good at; outside of the context of a form, it is far less likely for the desired element to be adjacent (or even close) in markup.

Since there are a variety of ways in which a novel pointing approach could be achieved, a study will likely be necessary to decide which is the most effective to pursue. A small group of users will be asked to provide feedback on the various proposed versions of the system, to see which is preferred.

3.2 How novel approach will be evaluated

This pointing system is intended to improve upon the pointing system offered by tabbing. The planned improvements pertain to lookup time for interface elements: that is to say, the number of key presses required to travel to some interface element from the user's initial state.

At this stage, some distinct cases of lookup to consider are:

- (From initial state) Lookup time of arbitrary interface element
- (From some focused element) Lookup time of element distant in markup
- (From some focused element) Lookup time of spatially distant element

- (From some focused element) Lookup time of semantically related element; for example, the next element in a list
- (From some focused element) Lookup time of semantically unrelated element; for example, traveling from a navigation bar to a content item

The system will try to support as many of these navigation types as possible. Evaluation will only be pursued on those navigation cases for which engineering is satisfactorily completed.

The system aims to be an effective, efficient and fast method for pointing. The system aims to compete primarily with keyboard-based tab-navigation. It is also intended to navigate interfaces that are designed for mice, so it is worth comparing its performance to mouse navigation also. Touch navigation deserves a mention, as it is an emergent pointing-based navigation method similar to the mouse. We chose to leave comparisons with touch-based systems out of the scope of this work.

Use of Metrics Metrics are required to evaluate whether the new system delivers on its goals, and to draw comparisons with the performance of existing systems. Care has to be taken to ensure the use of valid comparisons. Keyboard-based navigation methods can be easily compared with each other, since they can both be distilled to a sequence of keypresses. It is, however, harder to compare keyboard navigation to mouse navigation, as mouse pointing is done as a single continuous action (movement of the mouse) rather than a discrete sequence.

Efficiency Efficiency will be considered on a high level to be defined as:

$$\frac{\text{work input by user}}{\text{value gained by user}}$$

That is to say, how much effort is required to complete a task. Since the value to the user of a given navigation task is highly subjective, value shall be an arbitrary constant, 1. It must be warned that efficiency comparisons between differing classes of task will be invalid, since their relative 'value gained by user' is only really equal within their task class. In the case of keyboard, 'work input by user' can be measured in terms of how many inputs are required to complete a task. This also correlates to physical exertion demanded of the user, as it is a counter for how many inputs they need to make. The metric should not be used to draw comparisons against mouse efficiency (as explained in section 3.2). Since no convenient analogue for 'number of actions' exists for mouse pointing (only one action is required), efficiency could instead be measured in terms of how much effort is required in steering the mouse. Fitts' Law [15] can be used to model pointing times. Since the user is working for the whole duration, time in this case is a measure of effort.

The expectation set by Fitts' Law is that a necessary minimum amount of time will be spent in acceleration and deceleration to the target (the deceleration time being longer for smaller targets), and that longer journies take more time (because time is either invested covering distance at low speed, or increasing said speed and subsequently incurring a larger penalty in the deceleration stage). Care needs to be taken to minimize experimental error when measuring time, as the timer needs to be started and stopped on the right cues. Assuming a constant reaction time, this noise worsens (proportional to the signal) for shorter times. Thus steering experiments should be made long enough to minimize the error as a proportion, and/or repeated and averaged to reduce random error.

Speed Speed of the pointing mechanism will be defined as "time taken to complete task". This can be measured trivially, using a stopwatch (or more accurately by timers provided by the web technologies with which the user interacts). This is a metric that can be used to compare effectiveness of the novel system against tab-navigation and against mouse navigation. It can also be used to contrast the effectiveness of the three systems in each class of navigation. As explained in section 3.2, precautions need to be taken when designing and measuring experiments that hinge on time.

Effectiveness Effectiveness is the capability of completing a navigation task. In some cases it will be objectively impossible to complete a task (for example if the controls lack events for keyboard interaction, and keyboard interaction is the mode of navigation). In other cases it will be subjectively difficult; the user might not know how to use the system, or the system might work for the task but require an unreasonable amount of effort. In difficult or impossible trials, evaluating all the factors of the system's performance becomes more complicated (as time can tend to infinite, where the user aborts the task). User feedback is useful to articulate what the problems are in such situations. Participants can explain in writing whether (and why) the system is realistically useful for the anticipated pointing situations.

Learnability As users become experienced in using the system, their performance will converge on that of system experts. Their efficiency could improve (as they figure out how to minimize input work, or make navigation decisions that maximize output value), and/or their speed could improve (as a result of fewer actions or faster performance of said actions). This transition can be measured with subsequent measurements of speed or efficiency (on the same participant). Learnability of the novel system can be contrasted with learnability of the existing systems. The metric can also be used to ensure that users achieve some baseline level of competence with each system before measurements are taken (such that comparisons are fair).

A feel can be garnered for how 'intuitive' the system is, by observing users' first-time performance. However with the exception of the novel system, it might be hard to find first-time users of the other pointing systems, as they are mature and widely available.

Obstacle courses for the systems to compete on will be real-world examples chosen to embody well the navigation class at hand. Experiments can be workflow-based, and filled with enough tasks to give suitable confidence in the measurements.

3.3 User participation

Users will be needed for testing the performance factors of the system defined in section 3.2. Random selection of users will meet the needs of the anticipated experiments, except for where comparisons to expert performance is required. Users will declare their competence in factors that might be significant to test results. Identification of participants will not be required at any point.

4 Potential impact

If successful, this work will make web applications more accessible to users who can't (or won't) use the mouse as their pointing device. The software layer would create accessibility in websites which might previously have been difficult to use, or wholly unusable. For businesses serving web applications, that would otherwise be liable for excluding users (or would have to dedicate resources to remaking their websites), this would be a huge step forward.

It is hoped that the concept of 'pointing via key sequences' would be easily transferable to gesture sequences (in 2D/3D space), or spoken word sequences. This would allow the concept to have impact outside of the realm of accessibility, contributing to usability also – for example, in living room or mobile applications, where keyboard & mouse are not available for web browsing.

5 Roadmap

The system will go through the software lifecycle:

- detailed specification
- design
- development & testing

Then, the system will undergo study. Two studies are planned:
Study 1: Study of System Implementation Options

This study is to decide, should there be more than one option for how to make the system, which one is most well-received by users.
Study 2: Study of System Against Other Systems
This study is to evaluate the novel system and contrast it with the other navigation systems.

Each study will comprise:

- experimental design
- materials preparation for experiment
- experiment run
- results write-up

The roadmap for the project is described in fig. [1]. Should time pressures become apparent in the design/development phase, Study 1 could be omitted, with the navigation mode for the novel sytem then being chosen without user consultation.

6 Literature and Technology Review

The deliverable for this dissertation is anticipated to be a browser extension that adds accessibility onto web content, favouring a novel interaction approach over the experience prescribed as accessible by web standards.

The key points to explore are:

- What factors exist in accessible design for keyboard usage, in general and for the web?
- Have approaches to keyboard accessibility evolved over time?
- Has the status of keyboard accessibility evolved over time?
- Does literature support the notion that 'keyboard accessibility in the web is generally lacking'?
- Does work exist around making web content more accessible for the keyboard?
  - Has web accessibility changed over time?
  - Does work exist around adding extra accessibility to web content without the web developer's involvement?
    - Does this work aim to bring content in line with a suggested web standard, or is a novel approach preferred?

  - Does work exist around adding extra accessibility to web content via the user agent?

- Do there exist any novel keyboard pointing methods?
- Is there any support for the notion that web accessibility standards prescribe a sub-par experience?
- Are there any experimental techniques that are recruited for evaluating task performance on keyboards?
  - Are there any comparisons with mouse performance?

A review of each of these points follows. References to 'accessibility' in general will be focused primarily around keyboard accessibility. The search effort around 'keyboard accessibility' was constrained to 'pointing with the keyboard'; accessibility concerns around other keyboard roles, such as typing, will not be discussed.

6.1 What factors exist in accessible design for keyboard usage, in general and for the web?

It is useful to have an understanding of what is meant by 'keyboard-accessible' design. Deng, 2001 lists the factors involved [16] [as cited in 17]:

- Using a logical tab order (using the tab key from the keyboard to navigate from link to link) mapped to the layout of the controls and the layout of information on the screen
- Using keyboard mapping for speeding up keyboard interaction and enhancing alternative input methods
- Avoiding conflicts with the operation of assistive software such as screen readers, and exploiting the built-in accessibility features of operating systems
- Providing multiple methods for access via the tab key as well as the use of shortcut keys
- Defining hot keys for more functionality for example, allowing the user to go backwards from link to link
- Ensuring that access keys and hot keys for frequently used functionalities are reachable using one hand, for people using one hand only
- Avoiding repetitive key presses that would be uncomfortable for users with repetitive strain injuries

- Placing frequently used links and functions on the first navigation level without requiring the user to navigate a lot to reach them

This makes some good points (particularly that tabbing isn't the only way to expose functionality via the keyboard; shortcuts help too).

As for which issues were prevalent in practice, a list of 'Top 20' accessibility concerns was published in 2005, which included observations of the following keyboard concerns from accessibility tests [18]:

- Users cannot access objects by keyboard.
- Hot keys are confusing, missing, or conflict with browser commands.
- Focus does not move to the right place (so availability of target remains unknown).
- There is no visual focus on the page.
- Users cannot tab to page elements in logical order.

It must be stressed that this list is not exclusive to keyboard accessibility, and yet a quarter of accessibility concerns in this shortlist pertained to keyboard usage, and of those the majority were pointing issues. This suggests either that websites tend to be developed in a way that makes keyboard pointing difficult, or that the mechanism provided for pointing is not very effective.

6.2 Have approaches to keyboard accessibility evolved over time?

It would be interesting to explore the original role of the 'tab' key, and verify whether it is still honored today. However, its history proved to be ill-documented. Tab's nature suggests that it is for form traversal: there is no directional control, nor any promise of a particular destination beyond 'next element in markup' – these factors are not a problem in form navigation, where the intention is to travel to the next field. As such it seems well-designed for at least this, but the aforementioned shortcomings are much more relevant in pointing scenarios, where the destination is spatial, having more complexity than 'the next control', and can be unrelated to the starting position of the search. If it is the case that tab navigation was intended for form traversal, then it would not be surprising for it to be unsuited to general pointing. It is also easy to see how its use could have evolved from form navigation to whole-interface navigation; it provides a way to move keyboard focus to controls, so the temptation exists to purpose it as a general focus mechanism. On some level, this seems like accessibility, but on another level, it can be harmful; it pushes a possibly poorly-suited mechanism as the standard problem solution.

6.3 Has the status of keyboard accessibility evolved over time?

An early view (1995) is provided on the landscape of accessible computing by Bergman and Johnson [19]. It suggests (p.9) that "most interface style guides were not written taking users with disabilities into account". Whereas today, accessibility is featured in the Design Guidelines for many prominent software platforms: Apple references accessibility in its OS X Human Interface Guidelines[20], Microsoft provides literature on Designing Accessible Applications[21], and The GNOME Project provides Human Interface Guidelines for use with its desktop environment for Unix-like OSes[22].

Hendrix and Birkmire describe how, in 1998, the Mac versions of Internet Explorer and Netscape web browsers lacked any provision for selecting or activating web links without the mouse [23]. The Mac OS as a whole was also slated for not providing out-of-the-box support for accessing menus via the keyboard. These points are in stark contrast to today, where the default Mac browser, Safari, provides tab support by default[24], and the OS provides global keyboard shortcuts to access menus[25].

A landmark legal case in accessibility, Maguire v The Sydney Organizing Committee for the Olympic Games (SOCOG) [26, 27], 2000, made it clear that laws such as the Commonwealth Disability Discrimination Act 1992 (Cth DDA) could be used to enforce web accessibility, and that websites lacking provisions for the disabled could be defined as discriminatory, and thus be penalizable. SOCOG's defence, that the cost of accessible development constituted 'unjustifiable hardship', was rejected. Not only did the defendant have to redesign their website, they also had to pay damages to the plaintiff on top of this. This revelation that a lack of accessibility could be more costly than the perceived 'price' of accessible development.

6.4 Does literature support the notion that 'keyboard accessibility in the web is generally lacking'?

Literature suggests that "the current design of most Web sites makes ... efficient keyboard navigation nearly impossible" [28, p.1], and that "the design of ... web applications is highly optimized for users who navigate with a mouse" [28, p.1]. This could be due to lack of standards-compliance, as a 2004 study [29] revealed a large majority (81%) of websites fail to satisfy even the most basic checkpoints of the WCAG [6, 7]. The same study [29] showed through interviews that disabled users believed that most web sites of the time did not consider their specific needs. The problem of Web designers lacking knowledge of the requirements of disabled users has been reported for years [29] [30] [31]. This is understandable; the development of keyboard-accessible websites is considered to be challenging [11].

6.5 Does work exist around making web content more accessible for the keyboard?

As explored previously in section 1.3.1, guidance is provided on how to develop accessible web applications, in the form of the W3C's WCAG [6, 7]. However, the content itself is not the only factor; content is accessed via a user agent (ie a web browser), and the user experience of that agent is important too. The W3C provide recommendations on how to develop accessible user agents for browsing web applications[32, 33]. The User Agent Accessibility Guidelines recommends navigation methods for users, that could reduce the journey time for tab navigation. For example, Guideline 8 of the version 1.0 document, 'Provide navigation mechanisms', suggests that "User agents should allow users to configure navigation mechanisms (e.g., to allow navigation of links only, or links and headers, or tables and forms, etc.)" [32, p.17]. Certainly allowing a filter on which elements are navigated would reduce the amount of navigation actions. Indeed, as will be discussed in section 6.6, the Safari web browser allows the user to filter which types of focusable controls are traversed during tab navigation. However this filter choice is limited (only two modes are offered, and neither may be what the user really wants), and the feature is not equally available in other mainstream browsers (it is missing in Firefox, and Chrome's implementation buries it inside a preference [34]).

6.5.1 Has web accessibility changed over time?

Suggested standards have evolved, with two versions of the WCAG being published [6, 7], in 2001 and in 2008. Some noteworthy changes to keyboard use include [35]:

- The recommendation for provision of access keys has been redacted in the newer version.
- Focus traps (where the user can't move keyboard focus out of some interface area, such as an 'infinite-scrolling' list) are recognised to be a problem.
- Visual indicator of focus is now required for keyboard navigation.

Web accessibility as a research field is now growing quickly and increasing in diversity, despite few studies having been published until 2002 [36]. Freire, Goularte, and Mattos Fortes claim that this interest has been stimulated by a need for developers to address accessibility requirements. This chronology could perhaps be explained by the Maguire v SOCOG [26, 27], 2000 lawsuit referred to in section 6.3, after which accessible development gained some visibility.

6.5.2 Does work exist around adding extra accessibility to web content without the web developer's involvement?

In cases where the web developer is uninterested or uninformed about accessibility, they cannot be relied upon to introduce accessibility to their website. Kouroupetroglou, Salampasis, and Manitsaris presents a framework for annotating web-pages with semantic information about the role of content, to aid information access for disabled users of the Worldwide Web [37]. The paper is based on the idea of the Semantic Web [38], where structure and relationships of content are declared. The annotation framework relies on a community of users proposing markup for inaccessible web-pages, and uploading the annotation file to a public storage server. A bespoke web browser for blind users, SeEBrowser, loads the pages alongside their annotations, and uses the extra markup to provide structure-aware browsing shortcuts.

The advantage of divorcing the responsibility of accessible development from the web developers is that the onus is not imposed on someone who may have an incomplete understanding of the domain, and also that stakeholders are given the power to add accessibility themselves, should the official design be unsatisfactory. This is, of course, predicated on having an active community (since without annotations, no accessibility can be added). The approach may also be relevant to keyboard-accessibility, as blind users navigate via the keyboard. Certainly, providing shortcuts that allow a variety of manners in which to traverse content, gives keyboard users more control over their navigation. However, in mainstream browsers, the extra semantic markup would be less useful, as the only keyboard navigation method available is tabbing, which is considered to be inefficient even for well marked-up pages (discussed later in section 6.7).

6.5.3 Does work exist around adding extra accessibility to web content via the user agent?

Introducing accessibility via the user agent removes the need to change the way websites are developed. If existing websites do not meet content accessibility standards, the user agent may be able to compensate, by modifying the presentation of inaccessible content, or providing novel navigation methods that are robust to inaccessible page structure. For example, SeEBrowser (described previously in section 6.5.2) is a user agent that provides browsing shortcuts to aid blind users in navigating efficiently through various web page elements (such as content areas, navigational aids and functional elements) [39].

User agent customization has been used to modify Internet Explorer [40] to fit the needs of older users. This work aimed to aid vision impairments by zooming content and speaking text, aid cognitive impairments by simplifying layout, and help dexterity issues by providing large buttons, and an easy

interface to change keyboard settings. The reason it was chosen that the work be an extension on Internet Explorer was that users preferred to use a standard browser with accessibility adaptations, rather than a specialized one with a limited set of features. Content transformations were performed in the user agent, as opposed to the original design which proposed to transform pages in a proxy placed before the client (though this would make the featureset available on all web browsers, it was found to be a non-viable architecture [41, 42, 43]). User preferences were stored server-side, so that they would be obtainable from different computers. Ultimately it was found that most features offered were used, but the majority used were those that made only minor presentation adjustments.

A subsequent study built on this work by implementing a similar user agent transformation for Firefox [44], making the improvements available cross-platform. It describes the software as being purposed for changing the user experience, rather than the web page compliance.

6.6 Do there exist any novel keyboard pointing methods?

A patent exists [45] describing an 'intuitive' method of focusing elements using the keyboard. It is spatial in nature, and allows the user to specify the direction of focus travel via the keyboard (rather than leaving this decision to the arbitrary directional order prescribed by page markup). Whilst this improves over standard tabbing by allowing a choice of direction, it still does not give an indication in advance of what will happen after the keypress.

The default Mac browser, Safari, supports multiple granularities of interface tabbing, differentiated by whether Alt is held down with Tab[24]. This allows a choice of whether to navigate between all focusable elements, or just form elements. This essentially gives the user a filtering mechanism, and can shorten tab journeys where the only controls of interest are form controls (ie, forms). Chrome also provides two modes for tabbing, but they are accessessed via a setting toggle rather than given separate keybindings [34]. The result is that switching between them on a whim is not possible, so combining the use of the two modes in navigation is denied. The Mac OS itself also provides a global 'granularity' setting for tabbing, which can be toggled by shortcut between 'text boxes and lists only', or 'all controls'[25]. Again, the effect is that the tab journey can be made finer or coarser via a toggle, shortening journeys to certain elements. This system-wide toggle stacks with the effects of Safari's toggle, allowing for complex filters to be created. However, shortcomings are that the current state of the system-wide toggle has to be memorized to be able to predict effects, and as ever with tab navigation, the destination is unknown until after the action. Additionally, this setting is not honored by all browsers; the Chrome web browser, for example does not observe this preference[34]. As a solution, there are other impracticalities to consider: the setting affects interactions in all as-

pects of the OS interface, rather than just the application at hand, which may be undesirable. It is also questionable whether repeatedly changing system preferences just to complete navigation tasks within an application, is sensible from a design point of view; changes made in 'System Preferences' should be just that: preferences. A user can't 'prefer' both modes – rather, the choice is made based on current context, so a better implementation for these purposes is to allow both modes of navigation at all times, mapped to different key bindings (rather than toggling the mode used by one single key binding). In essence, this is what is achieved by Safari's 'Alt-Tab' shortcut.

MouseKeys is a feature that exists in Windows, Mac OS X and X Windows-based workstations. This is a system-level alternative pointing mechanism, that moves the mouse cursor and performs clicks, using keypresses. Naturally, being system-level allows it to work for all applications, and being an interface for operating the mouse cursor allows it to be treated by applications in the same way a mouse is. However it is considered time-consuming in comparison to keyboard navigation because it provides 'relatively crude directional control', and inefficient support for continuous motions like drag & drop [19].

Switch Scanning is a paradigm whereby input options are scanned in front of the user, who activates a switch whenceupon their desired option cycles into focus[23]. By allowing one key to perform many roles (dependent on time elapsed), it increases the expression of a keyboard shortcut. It is described as slow (naturally; it introduces a time factor to an otherwise-instant action). However, where time is not a concern, it could be harnessed to reduce the repetitive motor strain associated with repeated keyboard tabbing (a user could initiate cycled tabbing with one keypress, then stop it with one more, regardless of distance). For the purposes of this work though, which aims to improve on the speed of tabbing (not just the repetitive strain), switch scanning is likely too slow to be a solution.

TypeAheadFind (now Find As You Type), referred to previously in section 1.3.3, is not just a plugin for the Chrome browser, but also a feature distributed with Firefox as standard [46]. It streamlines for the user the process of selecting hyperlinks; any time keyboard focus is not within an input field, key input is directed to an as-you-type search, which focuses any matching string in the page. A selected hyperlink can easily be followed by pressing Enter. This is an efficient method for focusing hyperlinks, as text search can narrow down the target quickly, and without aiming. However the string needs to be sufficiently unique, to reduce cycling through options. Additionally, it is no use for selecting interface elements that are not defined by text, such as divs. It can also be a bad citizen if the web application in question expects to receive keyboard input when the user is not within a form (for example, in a game). This can be improved optionally by setting a single, uncommonly-used key to be used for invoking the feature.

6.7 Is there any support for the notion that web accessibility standards prescribe a sub-par experience?

Current accessibility guidelines do not address the problem of efficient keyboard access [28]. Keyboard inefficiency can be severe enough that even a standards-compliant website may be, for practical purposes, unusable by keyboard users[28, 30, 47]. However, as discussed previously in section 6.5, current user agents do not necessarily comply with user agent standards as wholly as they could; the problem could therefore lie with the existing implementations of web browsers, rather than the standards prescribed.

6.8 Are there any experimental techniques that are recruited for evaluating task performance on keyboards?

Schrepp, 2006 models keyboard performance using the Goals, Operators, Methods, and Selection rules (GOMS) technique described by Card, Moran, and Newell [48]. It is used for predicting how long an experienced user needs to complete interface tasks. The technique has been used to compare efficiency of mouse and keyboard techniques [48, 49]. An overview exists of the different GOMS models [50]. GOMS reduces human-computer interaction to the sum of elementary actions (either motor, cognitive or perceptual). For example, pressing of a button, moving a cursor to a target, or perceiving the current position of the mouse cursor. Schrepp freely admits that some of the factors in keyboard navigation are hard to model with GOMS; accounting for the cognitive demand of orientation (that is, ascertaining the current position of keyboard focus) is challenging. Though measurements for this time penalty are known (1.35s) [51], the frequency with which this penalty is incurred is harder to estimate (in fact, no guess was ventured on this by the paper). The ultimate conclusion was that, irrespective of this unaccounted extra time penalty, mouse speed still far exceeded keyboard speed. This further supports the notion that there is a wide discrepancy between mouse and keyboard efficiency. However the author concedes that application of GOMS for disabled users was questionable to begin with, as standard predictions of motor time may be too optimistic; this has been estimated as high as 0.6s [52], rather than the 0.2s [51] predicted for able-bodied experts. For some cases, such as repetitive strain, it is possible that motor problems would worsen with use, making the time factor worsen as the experiment progresses. This increases the challenge of modelling times.

For the purposes of evaluating the planned deliverable (an alternative keyboard pointing mechanism), the main comparison to be made is whether the novel system improves upon tabbing; for this it suffices to count keypresses. This avoids the difficulty associated with modelling usage times for disabled users, since the number of keypresses is still a useful indicator of effort, which is agnostic of physical condition. Time-based comparisons

could still be made between keyboard and mouse, but perhaps selection of able-bodied users would help to increase the validity of the modelling.

6.9 Conclusions

Keyboard navigation of websites is a problem. Correcting the way people develop websites is challenging, and unrewarding; many websites don't observe standards (because the web developers don't attempt it, or even know how), and even those that do, do not necessarily provide an ideal experience. This is not necessarily a fault of the content guidelines; rather, it seems to be the fault of the navigation method prescribed by user agent guidelines, or the interpretation of said guidelines. A different user agent that provides more manners in which to traverse content with the keyboard, could be the real solution to accessible keyboard pointing. Annotations used by blind users could be used also by sighted keyboard users to inform of content structure, and by extension navigation possibilities. User agent extension has been pursued before as a solution in favour of other mechanisms (such as content modification by proxy server). Thus it seems like the original proposal of augmenting keyboard pointing in the user agent via a browser extension, is indeed an approach worth pursuing.

7 Figures

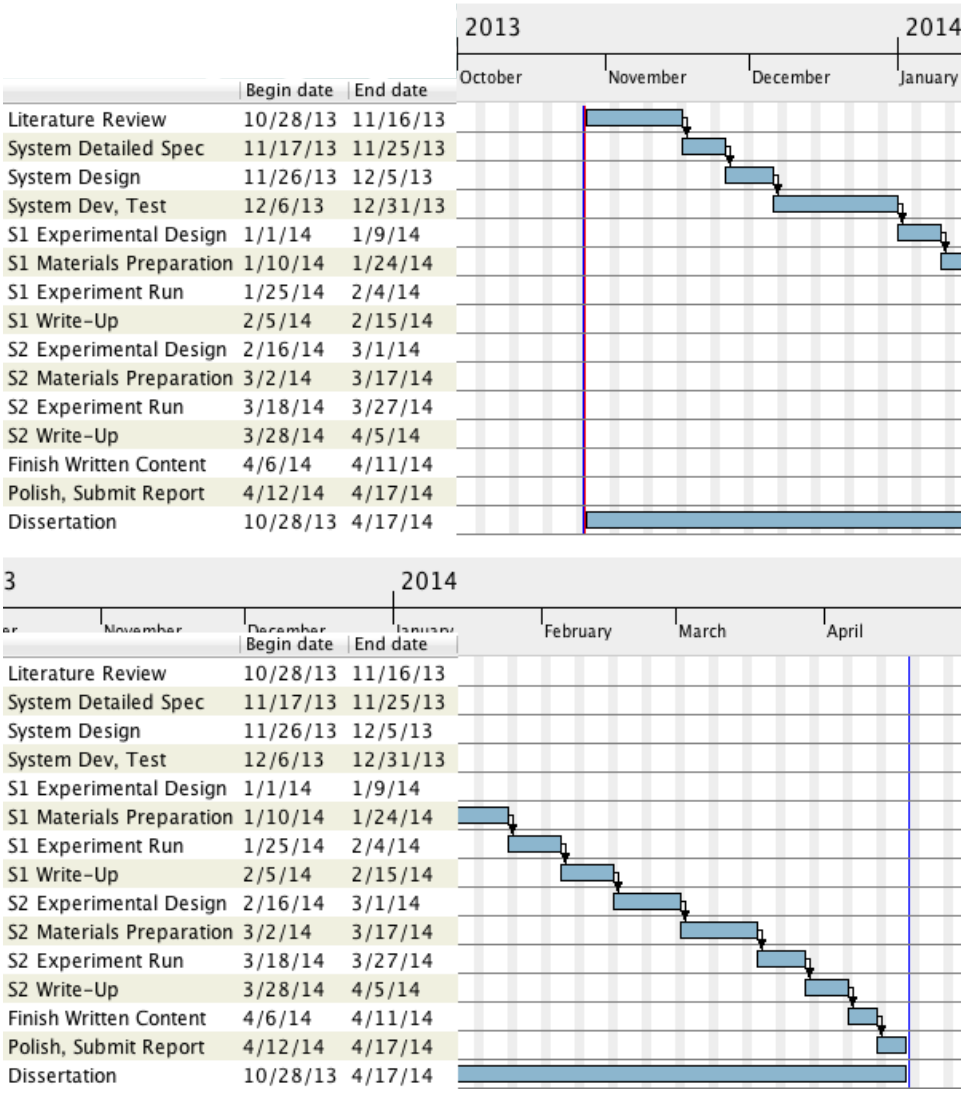

Figure 1: Roadmap of Work

8 Terms

access keys a means for users to travel between distant or unrelated interface components using keyboard shortcuts. 5

Cth DDA Commonwealth Disability Discrimination Act 1992. 16

GOMS Goals, Operators, Methods, and Selection rules. 21

key binding mapping of keys, or combinations thereof, to actions. 5–8

RSI Repetitive Strain Injury. 3

SOCOG The Sydney Organizing Committee for the Olympic Games. 16, 17

W3C World Wide Web Consortium. 5, 17

WCAG Web Content Accessibility Guidelines. 5, 16, 17

web browser extension (in the context of the Google Chrome web browser): extensions are small software programs that can modify and enhance the functionality of the Chrome browser. You write them using web technologies such as HTML, JavaScript, and CSS.. 6, 8

References

[1] Shari Trewin and Helen Pain. "Keyboard and mouse errors due to motor disabilities". In: *International Journal of Human-Computer Studies* 50.2 (1999), pp. 109–144.

[2] Eliana M Lacerda et al. "Prevalence and associations of symptoms of upper extremities, repetitive strain injuries (RSI) and'RSI-like condition'. A cross sectional study of bank workers in Northeast Brazil". In: *BMC Public Health* 5.1 (2005), p. 107.

[3] Ann E Barr and Mary F Barbe. "Pathophysiological tissue changes associated with repetitive movement: a review of the evidence". In: *Physical therapy* 82.2 (2002), pp. 173–187.

[4] Athula Ginige and San Murugesan. "Web engineering: An introduction". In: *Multimedia, IEEE* 8.1 (2001), pp. 14–18.

[5] Microsoft. *Outlook Web Client*. URL: http://www.outlook.com (visited on 11/17/2013).

[6] Wendy Chisholm, Gregg Vanderheiden, and Ian Jacobs. "Web content accessibility guidelines 1.0". In: *Interactions* 8.4 (2001), pp. 35–54.

[7] Loretta Guarino Reid and Andi Snow-Weaver. "WCAG 2.0: a web accessibility standard for the evolving web". In: *Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)*. ACM. 2008, pp. 109–115.

[8] WHATWG. *HTML5 Specification - User Interaction*. URL: http://www.whatwg.org/specs/web-apps/current-work/multipage/editing.html#the-accesskey-attribute (visited on 11/17/2013).

[9] Web Accessibility in Mind. *Keyboard Accessibility*. URL: http://webaim.org/techniques/keyboard/accesskey (visited on 11/17/2013).

[10] CabinetOffice. *Web page navigation*. 2002. URL: http://webarchive.nationalarchives.gov.uk/20100807034701/http://archive.cabinetoffice.gov.uk/e-government/resources/handbook/html/6-6.asp (visited on 11/17/2013).

[11] Willian Massami Watanabe, Renata PM Fortes, and Ana Luiza Dias. "Using acceptance tests to validate accessibility requirements in RIA". In: *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*. ACM. 2012, p. 15.

[12] Craig Campbell. *Mousetrap: A simple library for handling keyboard shortcuts in Javascript*. 2012. URL: http://craig.is/killing/mice (visited on 11/17/2013).

[13] tokland. *Type-ahead-find browser extension*. URL: https://chrome.google.com/webstore/detail/type-ahead-find/cpecbmjeidppdiampimghndkikcmoadk?hl=en (visited on 11/17/2013).

[14] userstyles.org. *Stylish browser extension*. URL: https://chrome.google.com/webstore/detail/stylish/fjnbnpbmkenffdnngjfgmeleoegfcffe?hl=en (visited on 11/17/2013).

[15] Paul M Fitts. "The information capacity of the human motor system in controlling the amplitude of movement." In: *Journal of experimental psychology* 47.6 (1954), p. 381.

[16] Y Deng. *Accommodating mobility impaired users on the Web*. 2001. URL: http://www.otal.umd.edu/uupractice/mobility/ (visited on 09/25/2005).

[17] Aspasia Dellaporta. "Web Accessibility and the Needs of Users with Disabilities". In: *KURNIAWAN, Sri; ZAPHIRIS, Panayotis. Advances in universal web design and evaluation: research, trends and opportunities. EUA, Idea Group* (2007).

[18] David Hoffman and Lisa Battle. "Emerging issues, solutions & challenges from the top 20 issues affecting web application accessibility". In: *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*. ACM. 2005, pp. 208–209.

[19] Eric Bergman and Earl Johnson. "Towards accessible human-computer interaction". In: *Advances in human-computer interaction* 5 (1995), pp. 87–113.

[20] Apple. *OS X User Experience Guidelines*. URL: https://developer.apple.com/library/mac/documentation/userexperience/conceptual/applehiguidelines/UEGuidelines/UEGuidelines.html (visited on 11/17/2013).

[21] Microsoft. *Designing Accessible Applications*. URL: http://msdn.microsoft.com/en-us/library/aa291864(v=vs.71).aspx (visited on 11/17/2013).

[22] The GNOME Project. *GNOME Human Interface Guidelines*. URL: https://developer.gnome.org/hig-book/3.10/principles-broad-userbase.html.en (visited on 11/17/2013).

[23] Paul Hendrix and Mike Birkmire. "Adapting Web Browsers for Accessibility". In: *Proceedings of Technology and Persons with Disabilities Conference*. 1998.

[24] Apple. *Safari Features*. URL: http://www.apple.com/uk/safari/features.html (visited on 11/17/2013).

[25] Apple. *OS X keyboard shortcuts*. URL: http://support.apple.com/kb/ht1343 (visited on 11/17/2013).

[26] Martin Sloan. "Web Accessibility and the DDA". In: *The journal of information, law and technology (JILT)* 2 (2001), pp. 01–2.

[27] Catherine Russell. "Access to technology for the disabled: the forgotten legacy of innovation?" In: *Information & Communication Technology Law* 12.3 (2003), pp. 237–246.

[28] Martin Schrepp. "On the efficiency of keyboard navigation in Web sites". In: *Universal Access in the Information Society* 5.2 (2006), pp. 180–188.

[29] Disability Rights Commission. *The Web: Access and Inclusion for Disabled People; a Formal Investigation*. TSO Shop, 2004.

[30] Kara Pernice and Jacob Nielsen. "Beyond ALT text: Making the web easy to use for users with disabilities". In: *California, USA: Nielsen Norman Group* (2001).

[31] Carlos A Velasco and Tony Verelst. "Raising awareness among designers accessibility issues". In: *ACM SIGCAPH Computers and the Physically Handicapped* 69 (2001), pp. 8–13.

[32] Ian Jacobs et al., eds. *User Agent Accessibility Guidelines 1.0*. Available at http://www.w3.org/TR/UAAG10/. W3C Recommendation, 2002.

[33] J Allan et al., eds. *User Agent Accessibility Guidelines (UAAG) 2.0*. Available at http://www.w3.org/TR/UAAG20/. W3C Last Call Working Draft, 2013.

[34] Todd Kloots. *Enabling Full Keyboard Access on the Mac*. URL: http://yaccessibilityblog.com/library/full-keyboard-access-mac.html (visited on 11/19/2013).

[35] W3C. *Comparison of WCAG 1.0 Checkpoints to WCAG 2.0*. URL: http://www.w3.org/WAI/WCAG20/from10/comparison/ (visited on 11/21/2013).

[36] Andre Pimenta Freire, Rudinei Goularte, and Renata Pontin de Mattos Fortes. "Techniques for developing more accessible web applications: a survey towards a process classification". In: *Proceedings of the 25th annual ACM international conference on Design of communication*. ACM. 2007, pp. 162–169.

[37] Christos Kouroupetroglou, Michail Salampasis, and Athanasios Manitsaris. "A Semantic-web based framework for developing applications to improve accessibility in the WWW". In: *Proceedings of the 2006 international cross-disciplinary workshop on Web accessibility (W4A): Building the mobile web: rediscovering accessibility?* ACM. 2006, pp. 98–108.

[38] Tim Berners-Lee, James Hendler, Ora Lassila, et al. "The semantic web". In: *Scientific american* 284.5 (2001), pp. 28–37.

[39] Christos Kouroupetroglou, Michail Salampasis, and Athanasios Manitsaris. "Browsing shortcuts as a means to improve information seeking of blind people in the WWW". In: *Universal Access in the Information Society* 6.3 (2007), pp. 273–283.

[40] Vicki L Hanson and Susan Crayne. "Personalization of Web browsing: adaptations to meet the needs of older adults". In: *Universal Access in the Information Society* 4.1 (2005), pp. 46–58.

[41] Peter G Fairweather, John T Richards, and Vicki L Hanson. "Distributed accessibility control points help deliver a directly accessible Web". In: *Universal Access in the Information Society* 2.1 (2002), pp. 70–75.

[42] Sara J Czaja and Chin Chin Lee. "Designing computer systems for older adults". In: *The human-computer interaction handbook*. L. Erlbaum Associates Inc. 2002, pp. 413–427.

[43] Vicki L Hanson and John T Richards. "Achieving a more usable World Wide Web". In: *Behaviour & Information Technology* 24.3 (2005), pp. 231–246.

[44] Vicki L Hanson et al. "Improving Web accessibility through an enhanced open-source browser". In: *IBM Systems Journal* 44.3 (2005), pp. 573–588.

[45] Jean et al. Mouyade. "Computer navigation method". Pat. EP2385452. Nov. 2011. URL: http://www.freepatentsonline.com/EP2385452A1.html.

[46] Mozilla. *Accessibility features of Firefox*. URL: http://kb.mozillazine.org/Accessibility_features_of_Firefox (visited on 11/20/2013).

[47] James J Powlik and Arthur I Karshmer. "When accessibility meets usability". In: *Universal Access in the Information Society* 1.3 (2002), pp. 217–222.

[48] Stuart K Card, Thomas P Moran, and Allen Newell. *The psychology of human computer interaction*. Routledge, 1983.

[49] Bonnie E John and David E Kieras. "The GOMS family of user interface analysis techniques: Comparison and contrast". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 3.4 (1996), pp. 320–351.

[50] Bonnie John. "Why GOMS?" In: *interactions* 2.4 (1995), pp. 80–89.

[51] Jef Raskin. *The humane interface: new directions for designing interactive systems*. Addison-Wesley Professional, 2000.

[52] S Keates, PJ Clarkson, and P Robinson. "Developing a methodology for the design of accessible interfaces". In: *Proceedings of the 4th ERCIM Workshop*. 1998, pp. 1–15.

9 Appendices

9.1 User confidentiality

This work responds to its institution's 13-point Ethics Checklist as follows:

1. **Have you prepared a briefing script for volunteers?** Briefing scripts will be provided to participants prior to any experiments. The purpose of the activities will be explained, as will be the destiny of the data produced by them.

2. **Will the participants be using any non-standard hardware?** As one goal of this work is to produce a highly-available system, hardware interaction is likely to be limited to keyboard & mouse work. Experiments involving gestures are not out of the bounds of possibility either, but no risks are anticipated with such interactions.

3. **Is there any intentional deception of the participants?** Deception will not be necessary for these experiments.

4. **How will participants voluntarily give consent?** Consent forms will be given to participants immediately preceding experiment participation.

5. **Will the participants be exposed to any risks greater than those encountered in their normal work life?** No; these experiments involve only safe human-computer interaction.

6. **Are you offering any incentive to the participants?** No incentive is expected to be offered to participants. Payment would not be used to increase the participant's risk of harm.

7. **Are any of your participants under the age of 16?** No.

8. **Do any of your participants have an impairment that will limit their understanding or communication?** No explicit selection will be made for such participants. The sampling will be random.

9. **Are you in a position of authority or influence over any of your participants?** No.

10. **Will the participants be informed that they could withdraw at any time?** Yes; this will be explained in the introductory script.

11. **Will the participants be informed of your contact details?** Yes; the contact details of the investigator and supervisor as part of the debriefing.

12. **Will participants be de-briefed?** Yes; an introductory script will be provided.

13. **Will the data collected from the participants be stored in an anonymous form?** Yes; data will be stored anonymously and securely.

NAME: Alex Bird

SUPERVISOR (IF APPLICABLE): _____

SECOND READER (IF APPLICABLE): _____

PROJECT TITLE: _____

DATE: _____