Accessible Pointing Approaches for Web Applications – Requirements Specification & Design

Alexander Birch

January 2014

Contents

1 Specification

Here we capture the main engineering goals of the software system to be produced. An attempt is made to refrain from prescribing a specific implementation.

1.1 Non-functional requirements

These requirements have a high priority:

R1 High availability
Neither price nor platform should be a barrier to adoption of the software. A solution that works in many browsers would be most available.

R2 Good citizenship (input)
Bindings reserved for interaction with software should not conflict with those required by web browser, or (within reason) bindings that webpage will use.

R3 Good citizenship (output)
Interface displayed by software should not prevent reading of webpage.

R4 Predictability
User should be able to anticipate the outcome of their actions in advance.

R5 Intuitivity (output)
User should be able to understand the output of the system.

R6 Intuitivity (input)
User should be able to understand what to input into the system.

R7 No time constraints
There should exist no time-sensitive interactions; user might be disabled in a way that makes it difficult to input quickly.

R8 Independent of markup
It is wishful to expect all visited websites to have sane markup, so semantics should not be relied upon. We point at the things that we can see, so a pointing system should be based around the visual layout of the page, rather than the semantics.

R9 Resilient to change
Many modern websites have changing interfaces, such as content being loaded in dynamically, or elements changing state (ie collapsible components).

Some of these requirements are inherently at odds with the choice to use a web extension hosted within the content of the page under navigation; necessarily, the interface of the software displayed in the webpage will need to overlap content on said webpage, and there is no telling in advance which keybindings will cause conflicts with an unknown webpage. The hope is that a solution can be found that works with a usefully large portion of the Web.

These requirements have a low priority:

R10 Co-citizenship with tabbing
Existing tabbing navigation should still be available to user, as it is still useful for some tasks (e.g. form navigation).

1.2 Non-functional requirements

These requirements have a high priority:

R11 Efficient navigation (semantically unrelated elements)
Navigation to elements that exist in separate visual containers to the currently focused element, should be possible in less than 4 keypresses.

R12 Efficient navigation (semantically related, but distant elements)
Navigation to elements that exist in the same visual container as the currently focused element, should be possible in less than 4 keypresses.

R13 Efficient navigation (arbitrary position)
Navigation to elements that exist in the same visual container as the currently focused element, should be possible in less than 4 lookups.

1.3 Anti-requirements

The following are explicitly *not* aims of the system, so performance in these areas need not be measured.

R14 Efficient navigation (semantically related, nearby elements)
Navigation to elements that exist in the same visual container as the currently focused element, and are nearby: tabbing navigation already performs this very efficiently (albeit unpredictably). The more pressing concern is how to navigate between visual containers.

R15 Navigation to off-screen elements
In this pointing-based interface, it is assumed that the user will only ever be trying to navigate to elements that they can point at; thus off-screen elements need not be captured.

R16 Fast navigation
The goal is 'efficient' navigation, not 'fast' navigation. There can exist a correlation (less inputs will, from a GOMS perspectively, necessarily

reduce task time also, provided cognitive and perceptual factors are negligible). But efficiency is the more important battle, as the system is targeted at low-bandwidth users, such as those with RSI (who need to avoid repetitive motor activity).

R17 Better performance than mouse pointing
This keyboard pointing method should be compared primarily to other keyboard pointing methods (ie tabbing navigation). It is for users who cannot use a mouse to point, i.e. due to physical reasons like disability, or interface constraints such as using a Smart TV Web browser requiring remote control input.

2 Design

2.1 Problem domain

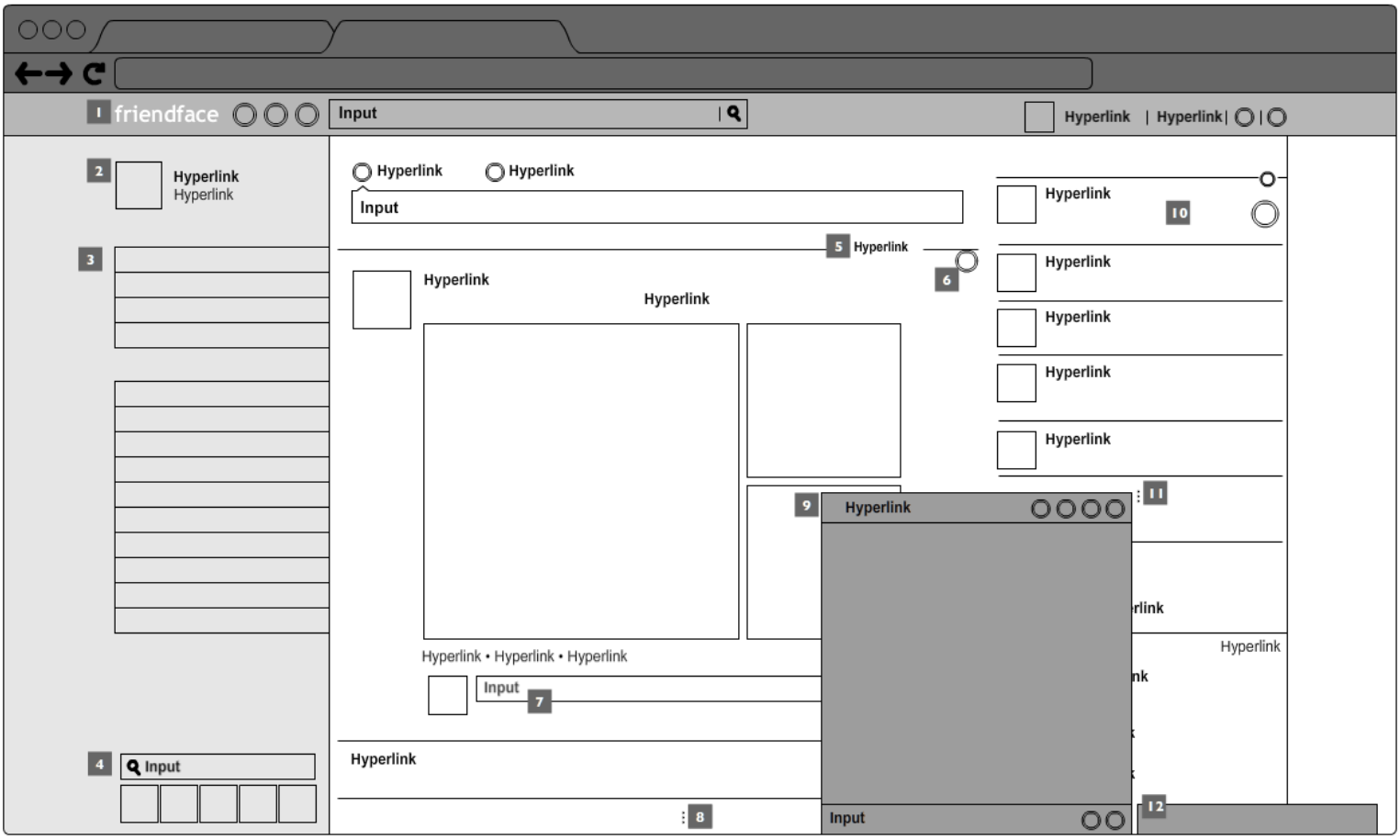

Figure 1: A typical complex web application layout.

Our software solution will need to cope with the many and varied pointing scenarios presented in typical web applications like Figure 1. Some discrete cases of interest have been labelled; a discussion follows, regarding what is significant about these cases, and how we could consider optimizing our software for such cases.

1 Navigation bar floats above content
This is a 'web application' version of interface 'chrome' (application controls around the content). Intuitive pointing needs to respect the current visual position of the content, and understand that the content can scroll under the floating chrome. Tab navigation ignores the presented position of the content, and can focus off-screen elements (despite the user's not being able

to see such elements). It would be more efficient to restrict the search space to visible, on-screen elements (i.e. things that can be 'pointed' at).

2 Elements can duplicate each other's actions

For example, a hyperlink could be the caption to some icon, and clicking either would produce the same result. In a poorly-marked-up page, a tab journey would traverse both of these elements, despite their function being equivalent. A more efficient mechanism would be one that allows the user to skip past duplicates (by combining them conceptually into one control, or otherwise by allowing the user to jump over multiple elements in a traversal).

3 Repeating units of repeating units

These sets of buttons are grouped into categories. If the user can express that they are interested in a category, it could eliminate other categories entirely from the search space. Similarly, if repeating units could be detected, then an explicit list traversal mechanism could be offered to the user. Detection of such things is likely to be difficult though, as it requires sane markup.

4 Unskippable lists

Similar to the previous point, a list of elements can be marked up without the use of an explicit list component (for example, a drop-down). In tab-navigation, skipping past the list is impossible; the user must tab past each individual element. A better traversal method would not rely on explicit list markup for entering and escaping lists.

5 Rarely-used elements

Though it needs to be possible to point at all elements, it is not ideal for obscure elements to appear in the majority of tab journeys (the user only wants to traverse them in exceptional cases). Our system needs to give the user control over the journey they take to point at an element, such that they can decide whether to traverse toward elements they would otherwise avoid.

6 Invisible elements

Some elements only present themselves upon mouseover or keyboard focus. Similarly, components like menus might expand as an option is touched on mouseover. Our pointing system might need to provide the user with an equivalent mechanism to 'mouseover' to probe for invisible elements. In particular, actions other than 'click' should be made available for interacting with the environment.

7 Many forms

A page can contain many small forms that are unrelated to each other. Since tab navigation is relatively good at traversing forms, browsers like Safari allow the user to traverse just form components. But it is hard to be sure whether pressing tab will move to the next element in some form, or to a different form altogether. It needs to be clear to the user how far they will move when they interact, so a visual indicator of the constraints of the search would be useful.

8 Infinite Scrolling

Many websites with dynamic content recruit 'infinite scrollers' to procedurally load in more content as the user scrolls the page. These are dangerous to tab navigation, which traverses depth-first; once it is in an infinite feed, it cannot escape, since as focus approaches the 'end' of the content, further content is loaded indefinitely. Thus elements after an infinite content feed are rendered unreachable. This is another case where our traversal system will absolutely need to provide a skipping mechanism, or choice of route.

9 Layout Occlusion

Floating elements can occlude those elements behind them. Thus it becomes impossible to point at the occluded elements. Actually, tab navigation copes very well with this problem (as it traverses by markup, rather than visual position). Our system will not try to solve this problem though, since even mouse pointing suffers from this. Since websites are mouse-optimized, it is expected that they would avoid creating problematic occlusion scenarios, or at least make them reversible (for example, some occluding content can be collapsed).

10 Repeated units of varying size

Identification of repeating units based on a visual analysis might be difficult, as the visual difference can be non-trivial; some elements can be larger than others. At any rate, our system will not attempt anything as complex as computer vision for understanding page layout. Ideally the system should be designed to not need an understanding of the page layout (like mouse navigation, which simply points at coordinates specified by the user).

11 Multiple Infinite Scrollers

A page can recruit infinite scrollers for content other than just the main feed. Thus assumptions like 'any infinite scrolled content is likely to be the main focus of the page' can't be so easily made. The main takeaway here is that our system should not optimize interaction based on some imagined understanding of the purpose of the page structure; it could easily make incorrect assertions, thus would not be resilient to bad or obscure web design.

12 Multi-modal Elements

Elements can change size or state, for example collapsible components. Any representation the system stores about the layout of the webpage should be sensitive to this possibility of change.