NetSurf after 1.0 – where to from here?

John-Mark Bell

2nd April 2007

1 Introduction

As development of NetSurf's first official release draws to a close, now would seem to be the ideal opportunity to evaluate what has been achieved. Further to this, and assuming development is set to continue, it is wise to consider future directions for the project.

It must be stressed that the views contained within this document are those of the author alone - nothing is set in stone. This document aims to stimulate the debate surrounding the goals for future NetSurf releases. Criticism and new ideas are welcome. The selection of features and technologies in this document is highly subjective, especially in the discussion of non-core technologies. Additionally, a number of technologies are included for completeness rather than as sensible propositions.

It is also important to be aware that this document is deliberately vague on the subject of how the suggestions made within it might be implemented. That information is too specific for a discussion document.

2 Where we are now

The current state of NetSurf is the result of 5 years' development. This section provides an overview of where NetSurf has been successful and where there is still room for improvement.

2.1 What's good

2.1.1 Achieved project goals

The original mission statement for NetSurf was "to bring the HTML 4 and CSS technologies to RISC OS". In the current browser, it is fair to say that this aim has been achieved successfully. However, that original mission statement does not cover a large proportion of the fundamental feature set that is required to realise a usable web browser. NetSurf 1.0 contains support for much more than just the two, key, technologies mentioned above.

2.1.2 Good ability to render much of the web

In terms of its rendering ability (and thanks mainly to its CSS support) NetSurf is capable of displaying a large proportion of the web in the way it is intended. Sites which are well designed and conform to standards are handled particularly well.

2.1.3 Speed

NetSurf is fast. In terms of responsiveness to user actions, it far outperforms competing products. The layout and drawing engine is reasonably well optimised.

2.1.4 UI

A key feature of NetSurf is how well it integrates with the native OS. This is achieved by use of native tools wherever possible, along with adherence to platform HCI standards. NetSurf's RISC OS UI goes beyond this and is considered by some to be setting the standard for user interfaces on that platform. Features such as the status bar nested to the left of the horizontal scrollbar have made their way into other applications.

2.1.5 Documentation & Graphic Design

NetSurf is well documented. The user manual weighs in at over 10,000 words and is illustrated by many images. The documentation has been translated into 3 languages (with a partial translation into a fourth). Graphic design plays an important part in NetSurf; the branding of the browser, design of the website and user interface themes are all affected by this.

2.1.6 Portability & Overall Design

NetSurf is portable. It has been seen running on a number of different operating systems, on a number of different processor architectures. These include:

Operating systems:

- RISC OS
- Linux (various distributions)
- FreeBSD
- Microsoft Windows (via GTK, not native).

Processor architectures:

- ARM
- x86
- x86_64
- SPARC.

NetSurf has also been ported to the Nokia N770 handheld device.

The design of NetSurf is particularly modular – handlers for different types of content may be added trivially. Implementation of new user interfaces is simplified by the coherent API and separation of most implementation details into platform independent and platform dependent parts.

2.2 What's bad

2.2.1 Intolerance of syntactically invalid documents

NetSurf's HTML parser is particularly picky about syntactically invalid documents. Its error handling mechanisms are nowhere near as robust as those of mainstream browsers and differ significantly from those other parsers.

This results in NetSurf exhibiting strange layout behaviour when faced with some sites. With over 90% of the HTML documents on the web being syntactically invalid to some degree, NetSurf's behaviour in this area leaves a lot to be desired.

2.2.2 Speed

Although NetSurf is fast, there is plenty of room for improvement. SSL fetches are particularly slow on ARM hardware. Additionally, large documents appear to tax the browser's core engine rather more than would be desired. Complaints have been seen in the past that "it's not as fast as Fresco"; some of this speed difference can, however, be put down to the fact that the releases of Fresco in question have utterly no support for CSS at all.

2.2.3 Memory consumption

Another common complaint about NetSurf (though significantly less frequently seen in recent times) is that of memory consumption. For some, NetSurf uses too much memory (and releases too little back to the OS when it is idle). This is clearly an area for improvement.

2.2.4 Incomplete CSS2 support

NetSurf's CSS2 support is incomplete. The vast majority of major features are present but there is still a significant subset of functionality that is missing. For example, support for @charset/@media blocks, !important rules and pseudo-selectors is missing. To various extents, these affect NetSurf's ability to render pages as they are intended.

2.2.5 Layout engine missing functionality

The layout engine is missing support for some CSS layout-related properties. The most obvious of these include fixed positioning, vertical alignment and text baselining. The current layout engine is not designed to handle dynamic changes to the underlying document or its styling.

2.2.6 No DOM/JavaScript

The primary complaint about NetSurf's feature set is the complete absence of JavaScript from it. To a number of potential users, this issue renders the browser useless (at least in their minds, if not in practice). Coupled with the lack of JavaScript is the omission of a DOM implementation into which a JavaScript engine could hook. Although this is more of an implementation detail, it is a critical problem which must be solved, should JavaScript support be required.

3 The future

In this section, we provide an overview of the technologies which could be added to NetSurf in the future, identifying the benefits of doing so and any disadvantages which may be incurred.

3.1 Core Technologies

3.1.1 HTML4 and later

Enhanced support for HTML 4 and versions of HTML beyond that. The critical issues here are parsing the document source into a consistent internal representation and applying the correct layout rules to each element.

In considering the parsing problem, it is important to be aware that it provides the ideal opportunity to bring NetSurf's error handling in line with that of mainstream browsers. It also allows for the production of a parsing architecture which is capable of handling extraneous source data to be injected into the input stream (cf. document.write()).

Correct layout is important for consistent rendering and interoperability with other user agents. Many of the layout requirements for HTML elements are directly representable in CSS. There are a number of elements which have special cases, for one reason or another. Many of these are already handled by NetSurf's current HTML handling; there are omissions, however, and these should be fixed.

3.1.2 XHTML

NetSurf's current handling of XHTML is to map it directly to HTML and use the existing HTML handling infrastructure to handle it. This is not entirely satisfactory, especially at the parse stage, as XHTML imposes a different set of requirements upon UAs than HTML. The most prominent example of this is XHTML's parsing requirements – being an XML dialect, encountering an error in the input document should abort the parsing process. There also exist a number of XML-specific features which an HTML parser would not be able to handle (processing instructions, for example).

Adding support for this would involve use of an XML parser to parse the document into a DOM. Ideally, this DOM would have the same interface as that for standard HTML (i.e. DOM nodes are the same, no matter what the source). To achieve this probably means making use of a SAX parser and building the DOM tree ourselves.

3.1.3 XML + XSLT / SVG / MathML

NetSurf currently has no ability to handle generic XML documents (styled with CSS or, through XSLT, converted into (X)HTML + CSS). This extends to a lack of support for SVG or MathML (which are both XML applications). The former is becoming more prevalent on the web, due to native handling being included in at least one mainstream browser.

SVG and MathML may be embedded as a subtree within another XML dialect's DOM (e.g. an XHTML document may contain elements from the svg namespace). The impact of this is that scripting implementations are able to manipulate these DOM subtrees in the same way as they may operate upon the containing document's. To achieve this fully would require a great deal of work.

3.1.4 CSS

NetSurf's CSS support currently encompasses the vast majority of CSS1 and a great deal of CSS2.1. However, the parser has its limits and does need revisiting. Also, pseudo selectors and !important rules are entirely unsupported.

NetSurf currently has no support for alternate stylesheets and dynamic stylesheet switching. This is needed from an accessibility perspective if nothing else.

At-rules aren't supported particularly well at present (especially @media blocks, the contents of which get treated as if they are part of the containing stylesheet regardless of whether their media type is applicable to the UA).

To address these issues, NetSurf's CSS parser will need a rewrite. In order to support dynamic pseudo classes sensibly, the ability to match styles dynamically is required. While these things are being addressed, consideration should be made as to how support for CSS3 and future versions could be added (even if such support does not appear in the short term).

3.1.5 DOM

NetSurf currently discards the DOM tree generated by the parser after the box tree has been created. This makes the addition of scripting support to the existing browser almost impossible.

The DOM comprises a number of different components, of which the Core, HTML and Events collections are the most important from a scripting perspective.

Adding support for the DOM APIs can take two forms; the first is to veneer them over existing implementation details (i.e the DOM is simply a part of the scripting implementation); the second is to make the DOM API a core component of the browser's engine and use it throughout. The former option isn't available within NetSurf, as the information needed to be able to make the DOM APIs a veneer is not available.

The CSSOM is also missing from NetSurf at present and support for this would ideally be added in future. Any DOM support should target DOM levels 3 and 2, ideally, although support for DOM levels 1 and 0 will, to some extent, be necessary for compatibility.

3.1.6 JavaScript

JavaScript is the key feature missing from NetSurf and, for some, is the barrier to it becoming their first-choice browser. A JavaScript implementation is comprised of; an ECMAScript interpreter, a collection of classes exposing the DOM API, a collection of classes exposing the non-DOM API (Window, History, etc), and a collection of classes implementing other things.

The major effort involved in bringing JavaScript support to NetSurf is likely to be that of the class libraries (and ensuring appropriate sandboxing for JavaScripts).

3.1.7 Layout Engine

As it stands, NetSurf's layout engine is completely unsuited to dynamic document changes (either through CSS or scripting). Should support for dynamic CSS or scripting be added in the future, this shortcoming must be addressed.

The layout engine also omits support for a number of layout-related CSS properties. Ideally, these will be addressed in future.

3.2 Non-core Technologies

3.2.1 IDN/IRI

NetSurf currently has no support for Internationalized Domain Names (IDNs) or Internationalized Resource Identifiers (IRIs). At some future point, support for these technologies should probably be added.

3.2.2 Flash

For the RISC OS build, it may be desirable to add support for Flash, either through a plugin or by embedding support directly within the browser. From a cross-platform perspective, utilising a plugin would be better, as it would allow usage of whichever plugin is native to the platform.

3.2.3 Plugins

Plugins in general are an issue which requires addressing in the future. The legal implications of the Eolas patent have caused other browser vendors to implement a "click to activate" interface for embedded content (i.e. the embedded content is not displayed until the user has clicked to activate the plugin task).

3.2.4 Printing

Printing is currently fairly rudimentary and prone to failure in some situations. It would be good to address this in the future.

Future developments could include the addition of a print preview mechanism; allowing the user to see what will be printed (and perhaps select the areas they wish to print). Taking account of print stylesheets when printing would be a worthwhile addition.

Printing on RISC OS 5 is fundamentally broken. Addressing this may be done in one of two ways; make NetSurf fall back to printing only ASCII text; fix the OS' printer drivers. The former is hardly a desirable solution (and one which has been studious avoided in the past). Therefore, if at all possible, fixing the OS' printer drivers would appear to be the best solution.

3.2.5 Disc Caching

Caching of content to disc has long been contemplated but, as yet, remains unimplemented for all contents which are not images. It may be a desirable improvement to add caching support for non-image contents.

3.2.6 Development tools

To aid web developers, it may be a good idea to include some kinds of debugging facilities, such as a JS console, log of parsing errors, DOM inspector, and so forth.

3.3 User Interface

3.3.1 Improvements

The common user interface is reasonably feature complete. This section details a number of ways in which it could be improved.

Keyboard navigation Keyboard-driven web page navigation would be a nice addition, providing functionality which is present in other browsers. One way in which this could be achieved is to use the CSS outline property to dynamically indicate the currently selected link.

Kiosk mode It may be desirable to add a kiosk mode interface, where the content area encompasses the entire screen.

Auto-completion for forms NetSurf currently possesses auto-completion for the URL entry bar. It would be a useful enhancement to extend this to all form fields. There are obvious issues here relating to password entry, however.

3.3.2 RISC OS UI

Technology The current implementation of the RISC OS user interface is in a state of flux. It is partially built upon a collection of generalised utility code. It would likely be useful to the wider RISC OS community if this code was extracted from NetSurf and a suitable library created from it, perhaps licensed under a liberal licence, such as MIT.

Once this has been done, the remaining rough edges of NetSurf's interface implementation can be ironed out.

3.3.3 GTK UI

The GTK user interface is partially completed. There is a large amount of ancillary functionality missing, however; along with a number of rough edges. In future, it would be desirable to improve this situation.

3.4 Portability

NetSurf is designed in a portable manner. This section describes ways in which this portability could be exploited.

3.4.1 Native Win32 port

A native port to the Win32 platform has been asked for in the past. It might be reasonable to attempt this in future.

3.4.2 Native OS X port

Similarly to a Win32 port, a port to OS X might be a reasonable proposition, especially as GTK integration with OS X's UI isn't particularly brilliant.

3.4.3 Ports to anything else?

Ports of NetSurf to other platforms have been suggested in the past. There exists a rudimentary port of NetSurf to the Maemo platform used by the Nokia N770/N800 tablets. This port is a variation upon the GTK interface, which accounts for the differences between GTK on that platform and GTK on the desktop.

A port to BeOS/Haiku has been suggested in the past, although nothing appears to have come of this; it might be an idea to chase this up.

3.5 Release Strategy

NetSurf's current development methodology has been feature driven and a long period of time has elapsed before a release has been made. This is mainly due to the need to achieve a suitable level of support for each of the technologies which comprise a web browser. In version 1.0, we see this goal being achieved.

In future, however, it would be more appropriate to schedule more regular releases with fewer features added in each release. This is a compelling approach as it allows features to be targeted at specific releases, giving end users some idea of when things will become available. It also allows development effort to be focussed upon specific areas. In making releases more regularly, it forces periods of stabilisation into the schedule, resulting in a higher quality product.

It is important here to determine how a release schedule might work; will it be time based (i.e. a release is made every n months) or will it be feature based (i.e. a release is not made until a certain collection of features is implemented). A time-based schedule would ensure regularity of release, but does not guarantee any level of feature addition or improvement between releases. Conversely, a feature-based schedule would guarantee feature addition but not a regular stream of releases.

A time-based approach would also enforce a regular stabilisation period. This would mean that new features would have to be implemented in a branch and merged into trunk once ready in order to avoid a feature's lack of completeness blocking a release.

Perhaps the best approach is a hybrid of the two - where specific features are targeted at specific releases but this is permitted to slip if too long a period has elapsed since the previous release.

3.5.1 Suggested Plan

This section details a suggested plan for the future development of NetSurf. The focus is upon the core engine (defined as targeting 3.1.1, 3.1.2, 3.1.4, 3.1.5, 3.1.6, and 3.1.7) as that is where most of the functionality is missing at present. Other changes would fit around this as necessary.

Releases should be made approximately every 6 months (and no longer than 12 months should elapse between releases). This equates to approximately 5 months development time followed by a month of stabilisation for each release.

Here's a suggested release schedule with a set of features that would be targeted for each:

Version	Features
1.2	New HTML parser (CSS parser, if ready) + DOM core/html, port box construction
1.4	New CSS parser (if not ready for 1.2) + CSSOM, port layout engine
1.6	DOM events + new box construction code, port layout engine
1.8	New layout engine, capable of dynamic content changes
2.0	JavaScript

Note that this is very sketchy and unlikely to be founded in reality. For example, 1.2 & 1.4 may be merged, depending upon speed of progress, as development of the parsers is fairly orthogonal. Also, 1.6 & 1.8 may be merged, with a longer release schedule due to the amount of work involved and the impact of changing the layout engine.

4 Conclusion

NetSurf's current engine is coming close to the end of its design life. This can be seen in the way that dynamic CSS changes simply don't work, except on a very constrained level. This obviously extends to the impact scripting might have.

Therefore, the engine needs revisiting to address these issues. While we are at it, we may as well make it more robust and less buggy. The engine is not the sole area where improvements may be made. These may be addressed in parallel with the engine changes as, in the main, they are orthogonal.

Section 3.5.1 reflects my own priorities for future development. Stated explicitly, they are; HTML, XHTML, CSS, DOM, JavaScript, and the required layout engine changes. Others' priorities are highly likely to differ.