The Underweb

An alternative World Wide Web framework

August Black
University of California Santa Barbara
Media Arts & Technology
august@mat.ucsb.edu

Marko Peljhan
University of California Santa Barbara
Media Arts & Technology
peljhan@mat.ucsb.edu

ABSTRACT

Development of new features and improvements to the WWW come by way of a precarious competition between browser vendors and ad-hoc interactions among the semi-closed boards of standards committees. Because all browser vendors provide their browsers to users at zero financial cost, and the growing majority of browsers (Chrome, Firefox, Safari) are free software, the kind of competition amongst vendors can now safely be viewed as artificial and antithetical to the inherent ideals of a universal framework for networked communication. The creative and critical research presented in this paper examines the problem space this contention creates and offers partial solutions in the form of a new free software browser prototype, called the Underweb. The Underweb provides developers and users with potentially more *user-elegant* technologies, including the ability to not only read, but also to write, edit and publish in a global communication system without private third-party involvement.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems

General Terms

Theory, Design, Languages, Standardization, Human Factors

Keywords

Underweb, multimedia, networked communication, free software, standards, user-elegance

1. INTRODUCTION

The Web as I envisioned it, we have not seen it yet. The future is still so much bigger than the past. Tim Berners-Lee

As the World Wide Web (WWW) has slowly evolved from a hypertext environment into a dynamic application environment, much has changed in the perceived function of this global communications framework. The browser is now more like an operating system than a simple hypertext viewer. The Underweb offers a new framework, both theoretical and practical, for an alternative and more elegant WWW aimed at giving 1) greater flexibility in programming multimedia content, 2) greater extensibility of the browser, 3) the production tools to read, write, edit and publish online without third-party intervention, 4) graphical editing for entry-level production, and 5) viable methodology for future development of the browser itself without political friction from *ex-ante* standardization processes.

The application framework that we have developed is in an embryonic, but usable, state. It aims to provide a practical prototype upon which future research could be built. We call it the Underweb because of two main reasons. First, unlike current browsers and their up-and-coming HTML5 standard, it freely exposes the underlaying APIs to the web developer instead of wrapping them in a high-level, constrained, and mono-linguistic framework. Secondly, it has the ability to underlay - and not just overlay - the application layer of HTTP. In a similar way and for similar reasons of flexibility and efficiency that TCP and IP were split years ago, the Underweb aims to break up the HTML-CSS-Javascript triad in favor of making lower-layer programming and scripting languages first-class entities with markup and style languages on top. Instead of being tied up with dynamic rich media content, we believe this would allow the HTML development to focus more on semantic issues and the original ethos of hypertext.

While we argue for the construction of an alternative communications framework, our proposal should not be viewed as a complete re-write of the WWW. Rather, it should be viewed as a re-structuring, or re-framing, of the intent, purpose, and development methodology of the WWW infrastructure.

We first look at the varied and multi-dimensional problem space of the WWW from a socio-technological perspective. We then discuss the technical, functional, and aesthetic dimensions of our proposed solution.

2. PROBLEM SPACE OF THE WWW

The problem space of the WWW is multi-faceted and large, consisting of a field of interrelated forces and agencies. While the WWW already facilitates global communication of multimedia information in an ever-changing and indirectly governed manner, it could also do so in a more open manner that doesn't undercut the purported mission of the WWW as a universal communication framework. New development models could facilitate greater interoperability among users as well as provide a much broader range of aesthetic and expressive functionalities.

In this section we briefly discuss what the inherent goal of the WWW is, how it currently falls short, and why. We first start with a statement of general principles necessary for our argument. We follow this with a non-exhaustive discussion of various perceived problem areas and plausible solutions. We place special emphasis on the current state of browser manufacturing where all browsers are given away without cost. Additionally, according to various unofficial statistics, browsers with free software rendering engines (Chrome,Safari,Firefox) make up roughly over half of the market share.[1] The statistics also show a decline of Explorer, the one major proprietary browser left on the desktop market.[2]

2.1 General Principles

The inherent goal of the WWW is to be a universal space of communication. The W3C's mission statement says: "The social value of the Web is that it enables human communication, commerce, and opportunities to share knowledge." It further states that its goal is to "... make these benefits available to all people, whatever their hardware, software, network infrastructure, native language, culture, geographical location, or physical or mental ability."[6]. To achieve this, it must allow open access and participation in the system itself as well as in the regulation and definition of its infrastructure. Currently, and thanks to guiding principles of a royalty-free and patent-free[5] web set by the W3C, users of the web can participate in the system in an increasingly open and accessible manner. However, the regulation and definition of the WWW infrastructure takes place behind semi-closed membership boards. On one hand, these standards boards are open to participation with a fee. On the other hand, they are overwhelmingly influenced by power players with significant vested interest in defining the infrastructure.

Ideally, the desires and ambitions of the browser makers would directly correlate with the desires and ambitions of users and the majority of industry that depend on the WWW infrastructure. Historically, however, this idealistic view has not been the case, and can be seen in the confrontational browsers wars of the late 1990s as well as in the current direction of the HTML5 standard. This discord between browser vendors who compete for market share is antithetical to the implied goal of the WWW. It is also unnecessary and inefficient if all browser vendors seek to produce an identical product for no cost.

Simultaneously, the WWW and internet have enabled new methods of social production that were previously unthinkable. Catalyzed by the WWW, the open methods of Free Libre and Open Source (FLOSS) software development have become a tested and validated way of producing software for a variety of computing groups and tasks. GNU/Linux, for example, has been ported and used on virtually all hardware systems from large clustered cloud computing[3] to cell phones[4] to embedded systems[5]. It has proven effective not only for hobbyists and system administrators, but also for desktops[6], Hollywood[7], education[8], government[9], and crucial industrial operations.[10] Despite the challenge of building a sustainable economic model of production, the diverse methods of free software development have proven to be efficient, robust, globally scalable, and even profitable[2].

The WWW is also first and foremost a public good and civic space in a similar, but not the same, way that public roads, schools, and parks are. However, unlike public roads and parks, as well as unlike the internet itself, the WWW is not maintained in the public interest. Instead, it is tacitly governed by free market ideologies and a reliance on proprietary models of software. A major conflict of interest occurs where the ideology of private industry, mostly from the US, leads the development of a public, albeit virtual, utility. A more pragmatic and realistic view would take heed of the public and international scope of the WWW.

2.2 There are no public alternatives

Currently, the WWW is taken for granted as the mainstay of online communication. Most, if not all, developmental energy of global multimedia communication is focused on this one amorphous thing. New tools at the API and protocol level need to be explored to investigate the assumptions, intentions, and methodology of the WWW.[4] Competition does not come in the form of alternatives to the running system, but instead in the form of tiny incremental ad-hoc adjustments. These adjustments form incompatibilities that break the fluidity of communication between users. These browser incompatibilities are also detrimental and costly to a significant portion of industry that relies on this infrastructure to provide services. A more cooperative development methodology is needed, and we believe our prototype offers a fresh new perspective.

2.3 Lessons from the Internet

The altruistic motivation for developing the internet came from a very delicate mix of interests. While the main impetus for the internet may have been militaristic in nature and funded through governmental grants, its design comes mostly from academics. At the time engineers were developing the internet, there were also other competing and incompatible commercial networks such as AOL, CompuServe, and Prodigy. These commercial networks, however, "..were crushed by a network built by government researchers and

[1]`http://www.w3counter.com/globalstats.php` (Nov. 2011)

[2]`http://www.w3counter.com/trends` (Nov. 2011)

[3]`http://www.ibm.com/developerworks/systems/library/es-linuxclusterintro/`

[4]`http://code.google.com/android/`

[5]`http://www.gumstix.com/`

[6]`http://www.ubuntu.com`

[7]`http://www.linuxmovies.org/`

[8]`http://laptop.org/en/`

[9]`http://arstechnica.com/open-source/news/2009/03/french-police-saves-millions-of-euros-by-adopting-ubuntu.ars`

[10]`http://www.linux.org/info/linux_industry.html`

computer scientists who had no CEO, no master business plan, no paying subscribers, no investment in content, and no financial interest in accumulating subscribers."[7, p.7]

A likely catalyst for this altruistic development of the internet came from government-sponsorship of the project and the academic standing of those involved. Without commercial incentive driving the planning and design of the internet, the engineers were able to pursue ideals of universal access and public participation and use. Additionally, since they were paid by research grants and institutions, they could work on it professionally, devoting themselves full time to its development. This sits in contrast to something like FIDOnet, whose genesis stems from a more hobbyist background and whose jerry-rigged infrastructure was fraught with technical problems of scalability.[7] Unlike the development of the internet, the lack of public institutional involvement in the development of the WWW has encouraged a coercive web environment built by industry in favor of private ownership of data where a cooperative and public environment is in fact demanded.

A new methodology for how to continue the development of the WWW in the interest of the global public is necessary. We propose a new framework that insists not only on a royalty-free and patent-free principles, but also on a copyleft free software licence of the infrastructural source code itself. A commitment to FLOSS as defined by the Free Software Foundation, would protect the civic and heterogeneous nature of this communication space. Furthermore, it would provide for greater, more fluid, and more direct user participation in the development of the infrastructure of the WWW, the most significant of which is the browser. This will, no doubt, require a paradigm shift in attitudes towards the social and cultural significance of software in academia.

2.4 Centralization

An unwritten goal of the WWW is to provide individual users with the means of producing and distributing their own content. The rise of Web 2.0, however, contradicts this goal where we see private services such as Gmail and Facebook bring proprietary centers into what would otherwise be a distributed and decentralized system. This also sits in contrast to other public centralized services such as Wikipidea which guarantee the accessibility of its data.

Centralization of the WWW into private third-party server systems is socially dangerous because it places too much informational and political capital into the hands of singular holders. There is a danger of building unbalanced and unchecked power structures where the precarious hold on the personal information of users forms an informational base for actuarial surveillance. It poses a cultural and ethical hazard to the general public who have come to rely on these central services, and who are targeted both with pinpoint advertising and cultural manipulation. Most importantly, it is also technically unnecessary if users have the production tools, the infrastructure, and the literacy to publish and maintain their own content.

Currently, there is little industrial and commercial support to create a truly decentralized and distributed system where users can easily create and maintain their own nodes in the network. This is evident in the APIs selected for inclusion in the HTML5 protocol that favor data consumption and exclude APIs that would give users the power of producing and publishing. See Section 2.10. Additionally, the kind of user-oriented tools that facilitate peer-to-peer production and distribution is threatening to the established commercial power structures that benefit heavily from intellectual property rights. It is very likely that there are more incentives to restrict and control the means of production and distribution than there are to support them.

Our suggested fix is to expand the focus in the APIs and protocols of the WWW to not only retrieve and view information, but also on editing and publishing it. At the very least, this would provide users with the necessary bare technical means.

2.5 Standardization

As it is, the WWW and its standards evolve through a battleground of competing browser developers and vendors. Users want more functionality, easy access, multimedia, and functionality for publishing their own content. Interoperability is also a key concern. Incompatibilities between browsers cost significant time and resources. This cost is shared by industry alike. Due to the increasing acceptance of free software, the kind of competition amongst vendors can now safely be viewed as artificial and misplaced.

Although the drive towards open standards in the WWW is commendable, there is little evidence to support the need for *ex ante* standardization if developers assume a stable FLOSS development model. The current recommendation for HTML5 is well over 125,000 words. As increasing complexity is introduced to the current standards, the pragmatic necessity of FLOSS becomes more apparent. It should be theoretically more efficient to produce one free software system with possible forks or dynamically loadable modules than it is to develop complex specs that are then implemented identically in separate products. In the specific case of the WWW, we propose that a stable FLOSS development model is the most appropriate way to achieve universality and interoperability at a global scale. Under FLOSS, the source *is* the standard to a significant extent. Official standardization would then occur *post hoc* as the use of specific technologies solidify.

Innovation is also a factor. Dropping the current standardization model would free developers to innovate future APIs and functionality without the unnecessary political constraints of artificial competition. To innovate in the format and protocol of the WWW, it is necessary to break standards. One tiny innovation, such as the XmlHttpRequest fuction, has the capability of changing the entire shape of the WWW. The adherence to standards makes the absurd assumption that there is a singular set way to exchange mixed and multimedia events over communication networks.[11]

2.6 Black-box technologies

The fact that the base exchange format of the WWW is open and viewable to all as a readable text formated protocol and language is a major contributing factor to the popularity and accessibility of the WWW. Without the ability to view, copy, and modify the source of WWW pages, the WWW would not have scaled as quickly or as broadly as it did. In contrast, black-box binary-only software technologies forbid users to view, study, or modify the inner workings of the soft-

[11]On the blog for the new HTML5, one developer mentions how (ex ante)"Standards are like sex; one mistake, and you're stuck supporting it forever!" `http://blog.whatwg.org/this-week-in-html5-episode-38` (Nov. 2009)

ware, thus restricting their ability to use the system fully. The prevailing black-box technologies in the WWW are Microsoft's Silverlight and Adobe's Flash. These technologies are detrimental to the civic space of communication in the WWW as they force users and developers to submit to the technological demands and regulations of private parties. These blackbox technologies are also completely unnecessary as no infrastructural software of the current WWW is sold for profit. Our Underweb protects users from blackbox technologies through GPL licencing.

2.7 Reliance on HTML

Of the three main client-side languages used to create WWW content, Javascript is the one with the largest space of design possibilities. In comparison, HTML and CSS become only convenience languages. Our proposed prototype gets rid of both HTML and CSS as the base format and style modifier, respectively. Instead, we offer multiple programming languages as a base upon which users can invent their own markup and styling languages that suit their specific programming intentions and methods.

2.8 Mono-linguistic

Javascript is currently the only supported scripting language in the WWW. Even though any language may be Turing complete, the syntaxes and semantics of these artificial languages provide developers with unique characteristics that are preferable to the developer for specific tasks. While limiting the development environment to one language may seem to provide desired constraints on an otherwise complex multifaceted software environment, it is too constraining and unnecessary.

Furthermore, in addition to weakly-typed interpreted scripting languages, the browser could also offer strongly typed low-level languages with direct or protected memory access. This would give developers more possibilities to innovate. Google's native client[12] is a recent and example of such development, albeit very complex and with the requirement of joining their services. If made universal and easy to program, this would lead to a very different space of communication and activity in the WWW.

Our solution to this problem of mono-linguism is to create a polyglot environment in which the browser allows for programming in multiple low and high-level languages. With new introspection technologies that allow scripting languages to automatically and dynamically bind to external APIs, binding to one language is about as easy as binding to another. Also, one object-oriented language we support is Vala. It has a very simple syntax similar to Javascript, but is as fast and diverse as C.

2.9 Audio/Video playback

After twenty years of development including multiple open source and royalty-free codecs, open and cross-browser audio-video playback is still not possible in the WWW. Without standardized and open AV playback, users must rely on blackbox technologies such as Flash and Silverlight plugins. Furthermore, even though there has been limited and disjointed progress to get simple media playback into the browser, it is still very unclear whether any consensus will be reached on what codec, if any, will be accepted. This casts doubt on the capabilities of the standardization process. Despite initial efforts to push Ogg Theora as the universal codec, passing it failed due to very speculative assumptions regarding submarine patents.[13]

Browser	release	Theora	H.264	Webm
IExplorer	9.0	no	yes	no
Firefox	5.0	yes	no	yes
Chrome	13.*	yes	no	yes
Safari	5.1	no	yes	no
Opera	11.5	yes	no	yes

Table 1: Listing of current browser support for the three main video codecs. A "yes" means the codec is supported by default without extra plugins.

Table 1 shows the lackluster cross-browser support for any of the currently proposed video codecs. With ongoing contention about the very basic and straightforward issue of video playback, user experience and personal expression on the WWW suffer. This points yet again to our solution which stresses a need for a switch to FLOSS development strategies where user rights come first. Our Underweb prototype offers built-in free software solutions for AV playback of practically all encoded formats.

2.10 Read, Write, Edit & Publish

Few, if any, browsers currently support the direct editing of WWW content. Even though developing the infrastructural tools for a writable format was at the very inceptions of the WWW[1], it currently lacks any real production tools for direct participation. In contrast to the initial intention of the WWW, this functionality is now supported mostly by centralized third-party services such as blogs, wikis, or social networking sites. This phenomena associated with Web2.0 has demonstrated the necessity not only for a writable web, but also for one where content is easily publishable. In order to give users the ability to act individually as a beacon of information in the WWW it will be first necessary to give them the basic technical tools. However, the current trajectory of the standardization process seems to favor APIs for reading and not writing or publishing in the WWW.

The HTML5 working group currently defines a number of new functionalities - some of them partially implemented in browsers - that provide users with new ways of viewing and interacting with WWW content. These include the canvas 2D drawing methods, drag-n-drop frameworks, cross-document messaging, microdata, and database functionality known as web storage. Another noteworthy API that the W3C is developing, but that is not part of the WHATWG or HTML5 group recommendation, is geolocation. Where there are decoders and client APIs in the new HTML5 recommendations, there are no encoder and no server APIs. New functionality for bi-directional communication are defined in the web sockets recommendation. However, it only exists as a client initiated service. Users are unable to create server sockets that listen for requests. Additionally, there

[12]`http://code.google.com/p/nativeclient/`

[13]`http://www.w3.org/TR/2009/WD-html5-20090423/video.html`

is growing demand for audio-video encoding and streaming that is demonstrated by the popularity of things like Google's voice chat and Skype. Providing missing APIs for server sockets, audio-video encoding, and media streaming could lead to new peer-to-peer production methodologies.

Furthermore, new computing power in mobile devices mixed with increased broadband accessibility are converging to create new yet untapped possibilities in publishing infrastructure. The possibility of building a more direct infrastructure is also increasingly plausible. New projects like the Freedom Box[14], that was initiated in 2011, already show a desire to for a more direct do-it-yourself participation in the WWW.

Our solution provides the basic APIs for users to run their own server applications as well as encode and stream audio-video. Additionally, our Underweb prototype provides built-in editing capabilities.

2.11 Literacy

One of the advantages of the original HTML format was that it was a fairly easy language to learn and use. However, even though HTML had a minimal vocabulary and can be considered much easier than contemporary HTML, it was also intended to come with a user interface editor so WWW users could plausibly read and write the web format.[3] This editing capability, however, fell to the wayside as new browsers raced to gain market share. Moreover, Wikis and similar overlay systems have proven to be more convenient for users.

The lack of technical capabilities in the system is not the only contributing factor to user-dependence on third-party centralized services for publishing. A lack of literacy is also a factor. A more technically elegant system would provide not only the basic tools necessary to publish one's own content online, but would also include a multi-tiered system with both entry-level and advanced technologies.

The solution to digital literacy is admittedly not an easy one to solve because of two main reasons. First, it is difficult to define what literacy means in these environments. Second, unlike the book or other literary formats, digital formats are flexible and ever-changing. Our rudimentary solution at the moment is a soft directed focus on designing a layout API that is both textually and graphically editable. See Section 3.3.

2.12 Unknown and multiple formats

The possible formats of communication on the WWW are only physically bound by the digital resolution and fidelity of display devices, loudspeakers, network bandwidth, computing power, and input mechanisms such as the keyboard, mouse, and tablet. These physical limitations, however, already offer a vast, almost infinite, space of possibilities to present and represent informational content. The logical and virtual formats of the WWW, as well as the procedural languages that drive them, help designers and developers search and navigate through the immense space of design possibilities. Currently, the core WWW specifications only provide a minimal diversity of these languages and formats. All innovation must happen on top of these limited base specs. New formats, such as SMIL, SVG, and VR have difficulty entering the system.

Our solution is simply to provide a framework that can load and install modules as needed, just like an operating system. Indeed, this method has already proven to be globally scalable in many existing free software operating systems. We refer the reader to the Debian apt repository system for just one example.[15] This kind of repository can only be both scalable and participatory under free software methodologies.

2.13 Rectangles

Because the original WWW was created outside of the graphic design world and was initially only based on text, the default layout mechanisms of HTML were blocky and rectangular, following a rather standard off-the-shelf newspaper-like layout. This legacy is still felt to this day. If you compare the current aesthetic of the WWW with posters, CD's and other print or digital media, the overall difference is immediately and visibly perceivable despite the addition of CSS and other functionality. The reason for this is twofold. The ongoing development of the WWW is adhering to HTML as the only first-class format, unnecessarily so. Secondly, even though current HTML can mimic any other 2D design exactly, the default behaviour of HTML is biased towards rectangular forms. This almost rigidly cubist awareness of layout guides the overall look and feel of the WWW.

Our solution is provide an initial layout library that can produce rectangular container shapes with embedded text just as easily as non-rectangular shapes. This layout library is also only one of many possible libraries a user could develop and use.

3. THE UNDERWEB

The main way the Underweb aims to deliver proposed solutions to the problem space of the WWW is through an *empty* frame and open framework. We say it is *empty* because it is built on already-existing technology and is meant to provide only a thin open layer for users to fill with their own functionality. In conjunction with that, we provide a very simple scene-graph library for drawing container shapes, time-based media, and text on screen in a manner in line with, but alternative to the way the current WWW lays out a page. These graphical methods mixed with other networking options forms the libmentiras internal library.

We first briefly describe the technical components used. We then discuss the Underweb framework and the accompanying libmentiras library below. We note to the reader that we have developed this project as a research prototype, not as a fixed product. All parts could be changed. Improvements are also necessary. We then discuss considerations and future work.

3.1 The Underweb technical makeup

The current Underweb framework is written in the Vala programming language[16] and is completely based on free software. It makes use of multiple GTK+ technologies, including GLib[17], GType[18], GObject[19], Gstreamer[20], and lib-

[14]`http://freedomboxfoundation.org`

[15]`http://wiki.debian.org/Apt`

[16]`http://live.gnome.org/Vala`

[17]`http://developer.gnome.org/glib/`

[18]`http://developer.gnome.org/gobject/stable/gobject-Type-Information.html`

[19]`http://developer.gnome.org/gobject/stable/`

[20]`http://gstreamer.freedesktop.org/`

peas.[21] We use GLib as a general-purpose cross-platform software utility library. It provides many useful data types, macros, type conversions, regular expressions, string utilities, file utilities, threading, messaging system, a main loop abstraction, reference based garbage collection, etc. GObject, and its lower-level type system, GType, are used by GTK+ to provide a portable object-oriented C-based API. It was also developed with the forethought of accommodating automatic transparent API bindings to other compiled or interpreted languages for cross-language interoperability. To that end, the additional GObject introspection library smoothly facilitates *lazy* cross-language bindings. Libpeas is a GObject-based plugins engine that provides extensibility to the Underweb browser. With libpeas and GObject introspection, developers can program applications for the Underweb browser in C, Vala, Javascript, and Python. C and Vala are dynamically compiled and loaded as shared objects using GCC and the valac compilers. Javascript and Python bindings are provided on the fly through GObject introspection and the Seed[22] and PyGObject[23] projects, respectively. Other languages are also theoretically possible, but not currently supported. Gstreamer is an extensive pipeline-based multimedia framework based on GLib and GObject that provides a robust audio-video back-end for both decoding and encoding of multimedia data.

Other C-based libraries are also used. Drawing operations are handled using the Cairo cross-platform 2D graphics library.[24] It has support for multiple output devices that include the X Window System, Quartz, Win32, image buffers, PostScript, PDF, and SVG files. We use Pango[25] for the layout and rendering of international text. It is also possible to extend the Underweb with any available free software libraries. Many of these software components are also the same ones used in the Webkit and Gecko rendering engines of Chrome, Safari, and Firefox.

3.2 The Underweb Framework

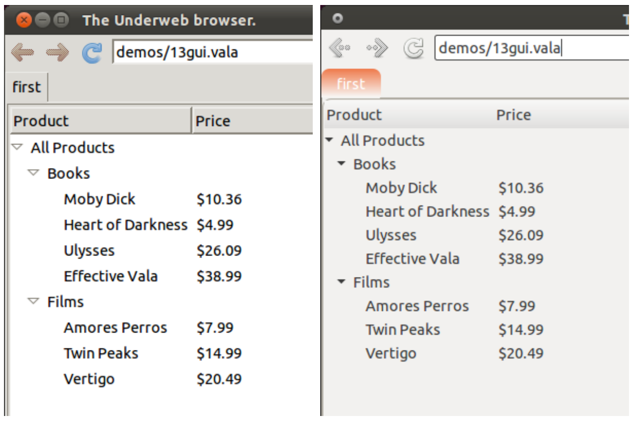
The Underweb provides a very flexible and thin pseudo-layer for potential WWW developers. It avoids the pitfalls of the current WWW by providing developers with tools that allow them to create application documents in many popular languages and using many available and professional free software libraries. In many ways, the Underweb framework brings functionality that was previously reserved for the base operating system into the realm of the browser. Furthermore, the Underweb framework can envelope and include already existing HTML-formated content using available free software APIs. We provide a library named libmentiras for handling graphical, acoustic, time-based, and textual data in a networked environment.

At the moment, the Underweb framework provides a simple graphical *frame* with a GTK+ window, a location bar and a tabbed sub-window. Like with other browsers, users type in the URI location of an online or offline application document, upon which the Underweb browser may then display and run the retrieved software. We call the viewable data of the Underweb "application documents" because they are neither full-on software applications or static documents.

[21]`http://live.gnome.org/Libpeas`
[22]`http://live.gnome.org/Seed`
[23]`http://live.gnome.org/PyGObject`
[24]`http://cairographics.org/`
[25]`http://www.pango.org/`

They are potentially a mixture of both. There is also an edit button that allows users to create and edit these application documents in the multilingual libmentiras format.

All application documents for the current Underweb are what would be traditionally known as a plugin. Based on the filename ending, either .py, .js, .c, or .vala, the Underweb browser loads the document and introduces it to the memory space it manages. The .py and .js files are interpreted by Python and the libseed Javascript engine, respectively. With .c and .vala application documents, the Underweb performs an extra step to compile and load the shared object files using the gcc and valac compilers.



Product	Price
All Products	
Books	
Moby Dick	$10.36
Heart of Darkness	$4.99
Ulysses	$26.09
Effective Vala	$38.99
Films	
Amores Perros	$7.99
Twin Peaks	$14.99
Vertigo	$20.49

Figure 1: The Underweb browser running a native GTK+ user interface example. The two windows are running the same application document, but under a different desktop theme.

The Underweb makes few assumptions about the kind of application documents a user may want to produce. It also has no limits on the various free software APIs that are potentially usable by the developer. For example, a developer is automatically able to produce applications that make use of any available GObject library, of which there are many professional-grade implementations of multimedia interfaces (gstreamer, cairo, poppler[26], Webkit[27]), graphical toolkits (GTK+, Clutter/Mx[28]), and networking(libsoup[29], GIO[30], d-bus[31]), etc. Furthermore, the Vala system also provides a simple way to bind directly to other non-Gobject C libraries. In this way virtually all modern free software APIs can be included in the Underweb, including openGl, sdl[32], couchdb[33], sqlite[34], postgres[35], pulseaudio, etc.

See Fig. 1 for an example that loads GTK+ widgets directly into the browser. The GTK+'s theming engine used by the Underweb is fully programmable with CSS styling,

[26]`http://poppler.freedesktop.org/`
[27]`http://webkit.org`
[28]`http://www.clutter-project.org/`
[29]`http://live.gnome.org/LibSoup` libsoup is an HTTP client/server library for GNOME.
[30]`http://developer.gnome.org/gio/2.28/`
[31]`http://www.freedesktop.org/wiki/Software/dbus`
[32]`http://www.libsdl.org/`
[33]`http://couchdb.apache.org/`
[34]`http://www.sqlite.org/`
[35]`http://www.postgresql.org/`

allowing the interface to mimic and tightly integrate with the graphical look and feel of the user's preferred operating system. A developer is not bound to use any external library, but with the ability of writing low-level C code can instead write computing intensive applications directly. The Underweb browser can in fact load, compile, and run iterations of itself in a truly autopoetic manner.

The Underweb exposes to the developer an object oriented interface for writing application documents. Each application document is essentially a software object as would be understood in Javacript, Python, and Vala. Besides object initialization which happens when the file is loaded by the Underweb browser, this object exposes hooks for two public functions that are called when the object is activated or deactivated. The relevant part of the base boilerplate code necessary for an Underweb application document is shown below in the Vala language. The syntax is different, although very similar, for Javascript and Python.

```
public class __UWEB__OBJ__ : GLib.Object,
 Mentiras.MyActivatable {
  public ScriptableWindow window {get;set;}
  public __UWEB__OBJ__() {   //initalize  }
  public void activate() {   //activate   }
  public void deactivate() { //deactivate }
 }
```

To develop for the Underweb, one would simply fill out the **activate** and **deactivate** functions with code that should be called when the application document is loaded or unloaded, respectively. Additionally, it would also be possible to provide users with the ability to pause a running application without unloading it to save CPU cycles. The **__UWEB__OBJ__** directive is a pre-parser substitution variable. When the Underweb retrieves an application document, the pre-parser then replaces all **__UWEB__OBJ__** directives with a SHA1 encoded hash digest based on the application document's full URI. This ensures that, internally to the browser, there are no namespace collisions between any two application documents.

The **window** object is available to all Underweb application documents in every language, and gives the developer access to the parent GTK+ widget. With it, a developer can add any GTK+ graphical widget to the display. At the moment, it only has one property, the **title**, with which you can set the display label of the current tabbed window.

One very useful library that the Underweb automatically hooks into is the free software WebKit engine. Webkit includes Glib and GObject bindings as a measure to improve speed in accessing the document object model of an HTML page.[36] This also allows the Underweb browser to seamlessly include regular HTML5 content inside of its visual frame, essentially making the current WWW a subset of the Underweb communication space.

Alternatively, an internal HTML renderer could also be written using the Cairo, openGl, or libmentiras libraries. Ideally, many markup forms would be invented and used that cater to specific design goals of various groups and individuals. Additionally, developers could invent their own template formats as are already popular in server-side programming environments. This would make WWW development easier to maintain and update by enabling developers and designers to work together more fluidly, and without the clunky and complicated server-side templates. By placing the communicable focus on the lower level programming languages instead of the higher level markup languages, we believe designers and developers are given more leeway and incentive to invent new high-level formatting in the client browser. This could include minimal and stricter markup and template languages that are suitable for beginners.

[36] http://trac.webkit.org/wiki/GlibBindings

3.3 Libmentiras

Libmentiras is a software library that aims to facilitate structured layout of multimedia content for Underweb developers. The libmentiras library provides a slim API for drawing simple curved shapes with wrapped text elements, full support for the reading and streaming (writing/publishing) of audio and video content, the support for TCP and UDP server and client sockets, a fully integrated HTTP server, functions for displaying text inline from URI, animation, and the loading and saving of files. Additionally, we have provided basic facilities for point-and-click editing and manipulating the shapes and textual contents in a libmentiras scene-graph layout of shapes. The syntax for creating these software items are very simple. They are also similar between the various languages, but will vary according to the binding. Because the framework of the Underweb allows developers to choose their layout software, libmentiras is only offered as one option. Developers are also encouraged to write and contribute their own layout libraries.

3.3.1 Graphics

The graphical layout engine of libmentiras uses a very simple schema for rendering graphics to screen. It consists only of a doubly-linked list of aggregated shapes. Everything that displays text or graphics, including time-based media such as video, is derived from a shape object. A layout with a single rotated and translated rectangular shape is created like so in Vala:

```
var layout = new LayoutEngine();
window.add_layout(layout);
layout.background_color={1,0,0,1};
var s = new Shape();
s.set_fill_color(1,0,0,0.5);
s.set_stroke_color(0,0,1.0,0.5);
s.stroke_width = 1;
s.rectangle(0,0,300,300);
s.set_text_from_uri(
        "http://underweb.info/text.txt");
s.set_text_color(0,0,0,0.7);
s.text_type = Mentiras.TextType.MARKUP;
s.shape_matrix = new Mentiras.Matrix2D();
s.shape_matrix.translate(100,50);
s.shape_matrix.rotate_centered(24,150,150);
layout.append(s);
```

In addition to holding and rendering shapes, the **LayoutEngine** object also has a settable property for background color. With a 4th alpha value, the background may also be transparent and could potentially offer new aesthetic possibilities that would allow Underweb application documents from separate URIs to be visually combined and layered. See Fig. 4.

The shapes are designed to be as simple as possible. Shapes consist of paths and their respective points, text, and a 2D matrix. The path may be closed or open. It may also be filled and stroked with color. The syntax will be very familiar to anyone who has done graphics development in other platforms. In fact, it only adds a thin layer of functionality on top of the libcairo graphics engine. Appending a shape to the layout puts the shape on the end of a list of shapes. When rendering is needed due to a change in any of the visible structure of the shapes, the shapes are drawn to screen in order, rendering new shapes on top of the previous shape.

All colors, including text and background colors are defined using 4 RGBA values. The vector path of the shape is determined by drawing commands such as **move_to** and **line_to**. The positions of points are given in Cartesian coordinates starting at the top left as (0,0) and going to the bottom right. Bezier curves may also be drawn using the **curve_to** function with three points defined as arguments.

The above code snippet also introduces the **Matrix2D** object. A matrix can be attached to the **shape_matrix** or **source_matrix** member of a shape, and can modify the positioning, rotation, scale, and general skewing of a shape's path or source. The source is any raster data such as an image, video source, or generic libcairo drawing context. We discuss this below in Sections 3.3.5 and **??**. The text also follows any matrix manipulation on the **shape_matrix**. If, however, the developer wishes to transform the points of the shape directly without matrix operations and without manipulating the text, she may also perform **translate**, **rotate**, and **scale** directly on the **Shape** object.

3.3.2 Text

Shapes also include optional textual components for formatting. A developer may choose to have plain text or one that includes a bit of markup. She may also choose to word wrap a text inside of the contour of the shape or within a rectangular bounding box. The **text_type** property of the shape object will set text rendering to accept either **MARKUP** or **PLAIN** text. The markup is defined by the pango library.[37]

The **text_fill_type** property of the shape object will word wrap the text inside of the shape (**FILL_WORD**) or inside of the shape's rectangular bounding box (**BOX**). Internationalization of text is also handled by libmentiras through the pango API with proper rendering of right-to-left and left-to-right scripts.

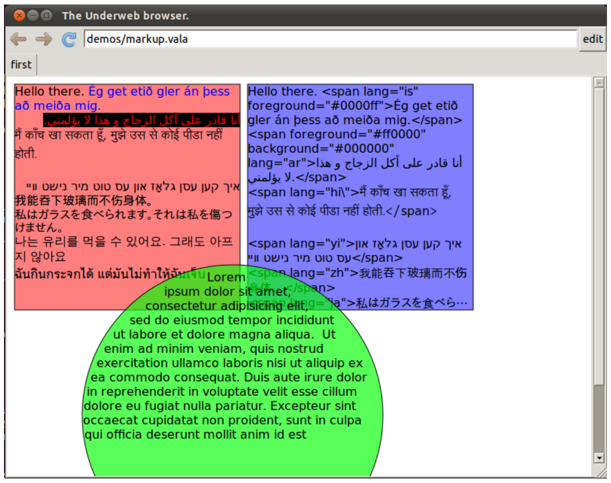
Unlike the current WWW, where text is encumbered by same-origin policy security restrictions, a libmentiras shape can safely load a text from an outside source without interpreting it as executable code using the **set_text_from_uri** function.

The **font_name** of a shape's text may also be explicitly defined, if not done so in markup. The **font_name** property accepts a string value in the form of **"[family-list] [style-options] [size]"**.

3.3.3 Interactivity and mouse events

Both the **LayoutEngine** and the appended shapes have event handlers for mouse input. The **LayoutEngine** has only three handlers for mouse move, mouse click down, and mouse click up. In addition to those three, shapes have mouse over, mouse out, and mouse drag. All events happen within the main loop of the Glib event system. To attach

[37]`http://developer.gnome.org/pango/stable/PangoMarkupFormathtml`



Hello there. Ég get etið gler án þess að meiða mig.
أنا قادر على أكل الزجاج و هذا لا يؤلمني.
मैं काँच खा सकता हूँ, मुझे उस से कोई पीडा नहीं होती.

איך קען עסן גלאָז און עס טוט מיר נישט װײ"
我能吞下玻璃而不伤身体。
私はガラスを食べられます。それは私を傷つけません。
나는 유리를 먹을 수 있어요. 그래도 아프지 않아요
ฉันกินกระจกได้ แต่มันไม่ทำให้ฉันเจ็บ

Hello there. <span lang="is" foreground="#0000ff">Ég get etið gler án þess að meiða mig.</span>
<span foreground="#ff0000" background="#000000" lang="ar">أنا قادر على أكل الزجاج و هذا لا يؤلمني.</span>
<span lang="hi\">मैं काँच खा सकता हूँ, मुझे उस से कोई पीडा नहीं होती.</span>

<span lang="yi">איך קען עסן גלאָז און עס טוט מיר נישט װײ"</span>
<span lang="zh">我能吞下玻璃而不伤身体。</span>
<span lang="ja">私はガラスを食べら…

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est

Figure 2: Red shape showing markup text. Blue demonstrating the same markup as plain text. Green shape demonstrating word wrapping inside a curved shape.

callback functions to them, a developer must connect a Glib signal to the function. In Vala and Javascript a developer may use anonymous functions to directly attach callbacks to these signals.

3.3.4 Animation and timing

Libmentiras timing is handled by the glib event loop. Thus, to create animations, developers may add callback functions to the event loop using the **Timeout** object. These methods are then called at regular intervals or with a timed delay.

Each **LayoutEngine** also has a member function that can check and update shapes in the scene graph at a regular predetermined interval. The **start_timer** function starts this timed method. The only argument it takes is an integer value designating the interval time in milliseconds. If a shape has changed in any way that requires a visual update, it then automatically draws it to screen. The **stop_timer** functions stops this procedure and removes the timer from the event loop. It is important to start a timer on any **LayoutEngine** that will have interactivity or time-based media. Without it, the layout will not be able to update itself automatically. Having this function as an optional feature instead of a default behaviour provides the developer with more fine tuned control over timing for resource-intensive applications.

3.3.5 Images

Images, like text, are a basic type in the WWW with high-level operations to load and display them to screen. In libmentiras, this is handled by the **Image** object. It is derived from the **Shape** class and includes a URI loading function that fetches, loads, and attaches pixel data to a shape's **source**. The example in Fig. 3 shows image handling with 4 images of different types loaded via HTTP in rectangular and non-rectangular container paths. SVG is also a valid image type. JPG2000 and other image formats will also be supported in the future.

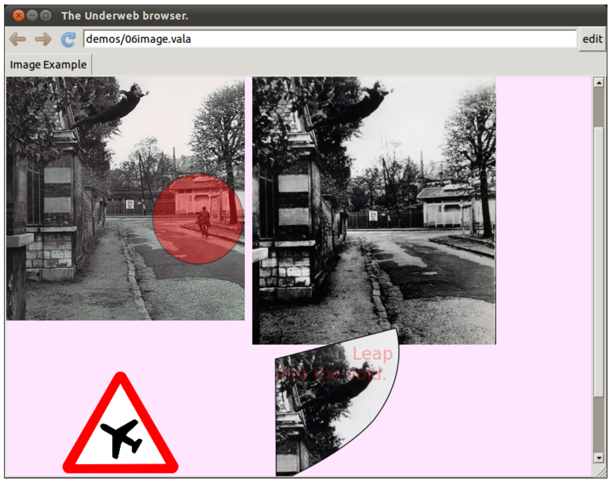
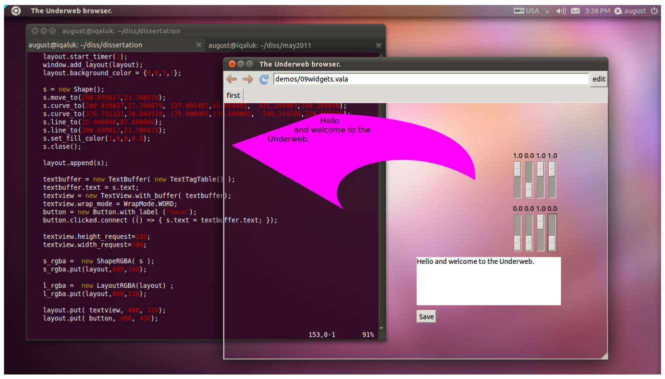


Figure 3: Example demonstrating PNG,TIFF, SVG, and Jpeg image loading and display.



```
layout.start_timer(2);
window.add_layout(layout);
layout.background_color = {0,0,1,1};

s = new Shape();
s.move_to(200.899017,21.706879);
s.curve_to(200.899017,21.706879, 537.001465,16.004593,  521.294983,160.169800);
s.curve_to(376.791321,70.843918, 179.000000,138.000000,  248.324280,220.597626);
s.line_to(15.000000,87.000000);
s.line_to(200.899017,21.706879);
s.set_fill_color(1,0,0,0.5);
s.close();

layout.append(s);

textbuffer = new TextBuffer( new TextTagTable() );
textbuffer.text = s.text;
textview = new TextView.with_buffer( textbuffer);
textview.wrap_mode = WrapMode.WORD;
button = new Button.with_label ("Save");
button.clicked.connect (() => { s.text = textbuffer.text; });

textview.height_request=100;
textview.width_request=300;

s_rgba =  new ShapeRGBA( s );
s_rgba.put(layout,600,100);

l_rgba =  new LayoutRGBA(layout) ;
l_rgba.put(layout,600,210);

layout.put( textview, 400, 320);
layout.put( button, 400, 430);
```

Hello and welcome to the Underweb.

Figure 4: Libmentiras widgets. Transparent backgrounds are also supported.

3.3.6 *Custom drawing*

The **source** member of a **Shape** object gives the developer access to a libcairo surface for fast and direct pixel manipulation. While the syntax for HTML5 canvas and libcairo are extremely similar, using the lower laying API of libcairo instead of HTML5's canvas API bypasses an extra layer of mediation. The would-be Underweb developer is also able to create an HTML5 canvas-style API internally on top of libcairo if she so desires.

3.3.7 *Widgets*

While HTML provides simple widgets such as text input and buttons, it has always lacked more intricate interactive widgets such as a slider.[38] Developers often created their own widgets with a mixture of Javascript and <div> tags. The Underweb browser gives the developer direct access to a broad range of lower-level GTK+ widgets, the presentation style of which is determined by system-wide settings and thus fully integrated into the look and feel of the user's desktop. The developer may also place these widgets directly inside of a libmentras **LayoutEngine**. Fig. 4 shows multiple slider widgets that control the color components of the shape on screen and the layout background. Additionally, a text box and button sets the shape's text. All widgets are placed within the layout using the **put** method of the **LayoutEngine** object.

3.3.8 *Video*

A major point of contention on the WWW has been video playback. Libmentiras makes it easy to play virtually any video formats from a URI or from disk using the Gstreamer multimedia framework. Like the **Image** object, the developer may place the video within an irregularly defined path. The **VideoPlayer** object also comes with a default set of graphical controls for play, pause, and seeking. When the user moves her mouse over video, the controls appear and become functional. The developer may extend or override these behaviours to suit her needs by deriving a new object from the video class.

Besides video playback, the Underweb can both encode and stream audio-video online, reading data from a camera and streaming it to an external Icecast server. Since the Gstreamer library is directly accessible to the developer through the Underweb framework, she may forgo or extend the rather simple high-level **VideoPlayer** and **VideoEncoder** objects to create custom media encoders and playback systems.

3.3.9 *Audio*

Consistent playback and synthesis of audio has long been absent from the WWW. While audio playback remains contentious and inconsistent among browsers, audio synthesis is just now making it into bleeding edge browser developments. Libmentiras presents a few methods for direct synthesis, manipulation, and playback of audio data. The basic hooks for a rudimentary audio synthesis engine are already in libmentiras. For computationally intensive media programming, the Vala language is more advantageous than the interpreted languages of the current WWW. Additionally, libmentiras supports the loading and control of PureData patches[39]. These patches can be used to build audio synthesis engines, among other things. A developer creates a Puredata engine by creating a **Pd** object, and then loading a patch through its **uri** property. Control can be sent to the patch from libmentiras through **receive** objects in Puredata. Other free software languages and media frameworks such as SuperCollider, Chuck, or Frei0r, could also be tightly and easily integrated.

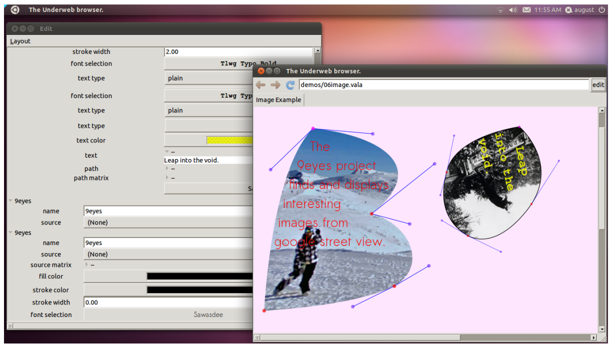
3.3.10 *Editing*

All Underweb application documents that use the libmentiras **LayoutEngine** may be directly edited within the browser. When a user clicks on the *Edit* button, a new dialogue in which she can create and edit shapes in the layout appears. This new dialogue provides the user with a point-and-click interface with which she can set all modifiable properties of a generic shape. Any and all shapes may be edited, including those like the **Image** and **VideoPlayer** objects that derive directly from **Shape**. The speed and re-

[38] A slider widget is in the HTML5 standards recommendation, but is not yet functional on the Mozilla 6 browser (August 2011).

[39] PureData is an open source graphical media programming language `http://puredata.info/`

sponsiveness of the browser is also fast enough to allow for the shapes containing videos to be edited while playing.[40] Fig. 5 shows editing operations in the Underweb browser.



stroke width 2.00
font selection Tlwg Typo Bold
text type plain
font selection Tlwg Typ
text type plain
text type
text color
text Leap into the void.
path
path matrix
9eyes
name 9eyes
source (None)
9eyes
name 9eyes
source (None)
source matrix
fill color
stroke color
stroke width 0.00
font selection Sawasdee
demos/06image.vala

Figure 5: Editing operations using libmentiras in the Underweb browser.

Paths and their respective points are also directly editable. The operations performed will depend on the mouse button and modifier keys. If a user clicks and drags any point with the left most button, the point will follow the mouse movement until the user lets go. If a user clicks a path point with the middle mouse button, that point will be deleted. A right click on any path point will iterate through the addition and subtraction of two control points. Additionally, the Ctrl button can be used to modify the behaviour of editing. Ctrl-clicking with the left mouse button on any path point will move the entire shape. Ctrl-clicking with the middle mouse button will scale the shape. Ctrl-clicking with the right button will rotate the shape.

3.3.11 Networking

Bare-bones networking features are also key in giving new functionality to the WWW that go above and beyond the projective recommendations of HTML5. Using the GIO and libsoup libraries, libmentiras is able to provide direct access to low-level software methods for serving and retrieving data over networks. The **ServerTCP** and **SeverUDP** objects of libmentiras can create basic TCP and UDP socket servers. The **Client** object can also connect to a server over a TCP or UDP socket. Attaching a message handler to the asynchronous **on_message** signal of either of these objects allows the developer to respond to incoming requests as they see fit and without interrupting the user interface. With these networking objects, developers can create their own domain-specific and user-generated protocols for peer-to-peer or mixed-use networks. Additionally, libmentiras provides a very basic HTTP server. In this way, an Underweb application document may serve another via HTTP. We encourage developers to write application documents that handle other protocols, such as Open Sound Control (OSC) or the Real-time Transport Protocol (RTP), or that create their own social networks outside of the available centralized systems.

[40]Depending, of course, on the video size and compression as well as the CPU of the user's machine.

3.4 Future Work

Our software is currently fast and responsive, but needs more features and development. Future work on the Underweb will need to focus on security models that don't inhibit design or usability, trust networks, built-in encryption, packaging systems, dynamic inclusion of foreign libraries via HTTP, faster and better text-wrapping algorithms, loading of web fonts, built-in streaming server, peer-to-peer protocols and methodologies for social networking. We also believe the API for libmentras is simple, but could be re-factored further to provide more simplicity and a faster scene-graph without loosing its current strengths. On top of it, new markup and style languages will be explored. Additionally, we would like to experiment with new URI schemes that not only point to a resource, but target specific rendering engines or versions of APIs (e.g. `http:webkit:3.2//underweb.info`). Furthermore, beyond a renewed academic investment in the development of the WWW, future sustainable free software development models based on economic justice for the programmers will also need researching.

4. CONCLUSIONS

We have discussed the current trajectory and problem space of the WWW. We have also presented an argument and novel prototypical browser that creates an open space for alternative developments within this complex field. We direct the reader to `http://underweb.info` for the downloadable source, videos, examples, and ongoing research.

5. ACKNOWLEDGMENTS

The authors would like to thank Dr. Curtis Roads and Dr. Rita Raley for advising this project.

6. REFERENCES

[1] BBC. Berners-lee on the read/write web. `http://news.bbc.co.uk/2/hi/technology/4132752.stm`, 2005.

[2] Y. Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom.* Yale University Press, 2006.

[3] R. Cailliau and C. Petrie. Robert cailliau on the www proposal: "how it really happened.". `http://www.computer.org/portal/web/computingnow/ic-cailliau`, 1997.

[4] G. Langlois, F. McKelvey, G. Elmer, and K. Werbin. Mapping commercial web 2.0 worlds: Towards a new critical ontogenesis. `http://journal.fibreculture.org/issue14/issue14_langlois_et_al.html`, 2009.

[5] W3C. W3c patent policy. `http://www.w3.org/Consortium/Patent-Policy-20040205/`, 2004.

[6] W3C. W3c mission. `http://www.w3.org/Consortium/mission`, 2009.

[7] J. L. Zittrain. *The Future of the Internet.* Yale University Press, 2008.