Redefine the Standard:
Design for the Transforming
World Wide Web

Takashi Okamoto
M.F.A. Graphic Design
Yale University School of Art, June 2004
Hon.B.Sc. Astronomy & Physics
University of Toronto, June 2002

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning, in partial
fulfillment of the requirements for the degree of
Master of Science at the
Massachusetts Institute of Technology
June 2008

© Massachusetts Institute of Technology, 2008
All rights reserved
Redefine the Standard:  
Design for the Transforming  
World Wide Web

Takashi Okamoto

Submitted to the Program in Media Arts and Sciences,  
School of Architecture and Planning, on May 9, 2008  
in partial fulfillment of the requirements for the degree of  
Master of Science at the Massachusetts Institute of Technology

Abstract

Elements within the physical space are transformed through natural  
processes that change their properties over time. These transformations  
are part of our everyday experiences that are often informative,  
influential or controversial. We are increasingly replacing interactions of  
the physical space to the Web, possibly losing opportunities for  
experiencing subtleties within these physical transformations. The goal of  
the thesis is to develop, understand and evaluate a framework for defining  
transformations on the Web; and establish a guideline to inspire visual  
and software design on the Web to leverage their socially influenced,  
time-based behavior.
Redefine the Standard:  
Design for the Transforming  
World Wide Web  

Takashi Okamoto
Redefine the Standard:  
Design for the Transforming  
World Wide Web

Takashi Okamoto
I like to start by thanking Kyle Buza and Luis Blackaller, two friends and colleagues in the Physical Language Workshop who together have gone through the highs and lows of the two years at the Media Lab. Thanks to all the friends I made during the past two years, especially my group mates: Kate, Mariana, Amber, Brent and Amna. Thanks to my parents for never telling me what not to do; and Hisashi, for being my one and only brother.

Thanks to my readers, Hiroshi Ishii and Natalie Jeremijenko for not only taking the time to read my thesis and provide valuable feedback, but inspiring me with their own work.

Finally, I am forever grateful to my advisor John Maeda. He is the reason I came to the lab, and the one who gave me the opportunity to do so. This is an experience I will remember forever.
Contents

1 Introduction .......................... 17
   1.1 Motivation .................................. 17
   1.2 Contribution .......................... 21
   1.3 Thesis Structure ......................... 21

2 Background ............................. 23
   2.1 The Web Platform ......................... 25
      2.1.1 Web 2.0 ................................ 25
      2.1.2 Networked Environment ................. 27
      2.1.3 Social Problem Solving ................. 28
      2.1.4 User Identity ......................... 32
   2.2 Information Filtering ..................... 33
      2.2.1 Data Types ............................ 34
      2.2.2 Search Engine .......................... 35
      2.2.3 Semantic Web ........................... 35
      2.2.4 Software .............................. 36
      2.2.5 Social Filtering ....................... 38
   2.3 Visual Systems ............................ 39
      2.3.1 Human Capacity ......................... 40
      2.3.2 Aesthetics and Graphic Design ............ 43
      2.3.3 Non-Photorealistic Rendering ............. 45
      2.3.4 Three-dimensional Information Landscape ... 46
      2.3.5 Visual Web Systems ..................... 48
3 Experiments

3.1 Web Applications ........................................... 53
   3.1.1 PLWire ................................................. 54
   3.1.2 OpenCode: Social Creativity and Education ....... 57
   3.1.3 GPC: Graphical Pen-based CAPTCHA .............. 61
   3.1.4 MudSketch: Web Drawing Interface ................. 63
   3.1.5 OpenLocker and FakeID: Web Anonymity .......... 65
   3.1.6 RunLog and RunLogger .............................. 71
   3.1.7 Implementation Summary ........................... 74

3.2 Visual Web System ......................................... 75
   3.2.1 E15 Overview ........................................ 76
   3.2.2 Visual Augmentation of Web Pages ................. 78
   3.2.3 E15:FB Visual Augmentation of Web Data ....... 84
   3.2.4 Implementation ..................................... 89

4 Discussion & Analysis ........................................ 93
   4.1 Analysis ................................................ 93
      4.1.1 Information as Entropy ............................ 94
      4.1.2 Age of Webpages ................................... 95
      4.1.3 Time-based Behavior .............................. 96
      4.1.4 Summary: A Transformation Framework .......... 102
      4.1.5 The Web Through E15 ............................. 104
   4.2 Evaluation .............................................. 108
      4.2.1 Participatory Design ............................. 108

5 Conclusion ..................................................... 111
   5.1 Conclusions ............................................ 111
   5.2 Future Work ............................................ 112
List of Figures

1-1 Andy Warhol, Oxidation Painting, 193 x 132.1 cm (76 x 52 in.)
Urine on metallic pigment in acrylic pigment on canvas, 1978.  18

1-2 Various works by Banksy. ............................. 19

2-1 DIMES data showing North American city-to-city connections [18]. 28

2-2 Examples of web applications in human computation. . . . . 29

2-3 The Sheep Market created artistic tasks for workers in Mechanical Turk. ............................... 31

2-4 sense.us collaborative visualization system, main interface components: (a) interactive visualization applet. (b) graphical annotation toolbar. (c) bookmark trail of saved views. (d) comments field. (e) comments for current view. (f) url of current state of the view. ................................. 32

2-5 Graphic annotation through drawing, commenting on the great depression with the accompanying comment "Great depression ‘killed’ a lot of brokers.” ................................. 32

2-6 Attribute-mapped scroll bars in various modes: (a) a normal scroll bar without wear; (b) snapshot of edit wear; (c) snapshot at a later stage with two categories of wear displayed; (d) showing total read wear and (e) read wear showing various categories. [21] ................................. 37

2-7 UML of source code, showing heat map of wear [11]. . . . . 38

2-8 The Model Human Processor. ............................. 41
2-9 Stroop Effect: Ignore what each word says and name the color each word is written in. Compare how much longer it takes for the right column compared to the left. ................. 42

2-10 Cave paintings from Lascaux. One of the earliest examples of visual communication, from 14,000 BC. ................. 44

2-11 Works of photorealism in American painting. ................. 46

2-12 Example of three-dimensional space occupied by large amount of text. Shown here, David Small’s Virtual Shakespeare Project. 48

2-13 David Small’s Virtual Shakespeare. Four views of the complete collection of William Shakespeare at different scales. .... 49

2-14 DataMountain showing 100 webpages in a 3D desktop environment. Users are allowed to organize webpages freely within the interface. ................................. 50

2-15 Example of inline hypertext: dictionary in Mac OS X. ...... 50

2-16 Hypertext: Pad++ for displaying graphical representations of hypertext. .................. .................. 51

3-1 PLWire, group webpage for the Physical Language Workshop. 55

3-2 OpenCode interface for the code editing page, showing the editor on the left and code repository on the right. ........ 58

3-3 Running an OpenCode program within the browser window. 59

3-4 Graphical pen-based CAPTCHA system. .................. 62

3-5 MudSketch, a drawing tool written in JavaScript. .......... 64

3-6 A couple of detailed images drawn using MudSketch. .... 65

3-7 OpenLocker showing various lockers throughout the MIT map. 67

3-8 Opening the locker requires users to unlock a combination lock. Inside the locker, users can create modules of their favorite RSS feeds, which they can move around the locker freely and reposition. .................. 68

3-9 The FakeID Website. ..................................... 69

3-10 The RunLog Website. .................................... 72

3-11 RunLogger, RunLog as a Facebook application. .......... 73

3-12 System diagram of a typical Ruby on Rails web application with three mongrel processes. .................. 75
3-13 E15 main interface components: (a) OpenGL-enabled 3D environment. (b) WebKit-based web browser. (c) Error console. (d) Python console. 76

3-14 Various visual filtering in E15. 81

3-15 Showing various levels of detail. Blurring decrease as user gets closer to element. 82

3-16 Drawing layer on top of E15's web browser enabling the user to highlight areas of interest. The aged webpage shows the highlighted areas. 83

3-17 Manipulating images and textual content of the New York Times website. Top story reads “The Pope Challenges President; To Arm Wrestling” 84

3-18 Showing current logged in user with friends. 86

3-19 Screen shots from E15:FB. 87

3-20 Facebook visualization of social interaction. 88

3-21 Searching google and showing results of friend’s favorite bands. 89

3-22 E15:Web in the E15 browser allows scripts to be loaded through web services. 91

4-1 Web counters show how many people accessed the current page. 95

4-2 MudSketch sketch loading window. 99

4-3 Inside an OpenLocker. 100

4-4 Aging effect over time, from left to right. Top row: 1 day, 7 days, 14 days. Bottom row: 30 days, 180 days, 365 days. 105

4-5 RunLogger users post comments and suggestions in a public discussion board. 109
1

Introduction

1.1 Motivation

Physical elements has a way of transforming over time. As with all things physical, its existence is impermanent and affected by environmental conditions. Patination is the first law of life, guaranteed by the second law of thermodynamics. The physical space is transformed through natural processes which occur over time. The inevitability of age, is something that many of us may be fearful; yet without it, the world does not feel real. At least that is what I have come to realize after spending more than half of my life creating digital work on the computer. As we spend more time in front of the computer and on the Web – replacing real world interactions with online social interactions – it is important that we bring the realities of the physical world into the Web.

There exist a rich culture around the appreciation of physical transformations. The Japanese philosophy of Wabi-sabi – contradictory to the hellenic view of the West which considers permanence, grandeur, symmetry
and perfection as preferable qualities – see intrinsic beauty in impermanence, humility, asymmetry and imperfection. Impermanence, by accepting the passage of time, simple and rustic qualities create innate beauty found in the flow of nature. Because these imperfections and complex characteristics are enjoyable, there are attempts to intentionally create imperfection. Japanese paper is produced with wrinkles rich in texture; unlike the smooth, bleached, perfect white paper we use conventionally. Having complex qualities is something we appreciate, or as John Maeda simply says, “We □ complexity.” [36].

Artists throughout history also embrace physical transformations. Andy Warhol experimented with new forms of painting. Unlike his mass-produced silkscreened paintings surrounding the core theme of pop art, his oxidation paintings [Fig.1-1] produced work which were more Pollock – more abstract and expressive. He painted copper-based paint on canvas, and had his assistants urinate on them which oxidized the copper to leave patterns of discolorations. This transformation left an irreversible mark that represented the motion of the painter. A rather obscene gesture is converted into something aesthetically beautiful.

Fig. 1-1 Andy Warhol, Oxidation Painting, 193 x 132.1 cm (76 x 52 in.) Urine on metallic pigment in acrylic pigment on canvas, 1978.
1.1 Motivation

Graffiti writers transform street walls into canvas. Their works are seen as art by some while others see it simply as an act of vandalism. There are many reasons as to why people paint street walls. To some, it is art, to others it is a communication device; and to most, it is just something to do. However, observers feel about the actual work, their actions raise an interesting question— who owns public spaces? Banksy, a graffiti artist from England, uses public spaces to exhibit his work of subversive political messages. Like many graffiti writers, his identity is secretive and unknown. His works are highly political and in many cases illegal. The work themselves are primarily created through spray paint and stencils. They are sensitive to location and often integrating preexisting elements in the space [Fig. 1-2].

Fig. 1-2 Various works by Banksy.

What happens to digital information, specifically in a public space like the Web? What happens to webpages over time? Because web information is not physical, a webpage that is uploaded to a server will look the same no matter how much time passes. The lack in transformation due to the absence of the second law of thermodynamics leads me to believe that we are losing opportunities for information to give additional clues, which may act as helpful aesthetic metadata for its audience.

The idea of Web graffiti exists in a way. Typically, webpages are read-
only. However, certain types of webpages, such as blogs and wikis allow users to modify or add to the contents. Wikipedia for instance is a source of constant controversy as anyone can rewrite entries within the wiki-based encyclopedia. There exists the power for anyone to modify the contents of an entry. Just like the physical world, a wiki writer can go to a wiki-page and modify it to their liking, overlaying their work on top of the current one.

In the past two years in the Physical Language Workshop, I have explored various aspects of the Web through the deployment of web applications. As web applications, each project becomes instantly accessible to everyone on the Web. This enables opportunities to create communities on the Web and create an environment for projects that require user participation and contribution, exploring the social, time-based behavior. The projects try to address problems or issues that exists on the Web, or take advantage of the networked nature of the Web environment to collect user contribution. Some examples of these projects will be outlined in Chapter 3.

The means of interactions with these web applications occur through the web browser, and the user interface is displayed through webpages. With background in exploring the Web through web applications, we at the PLW began to think about how we can interact with web content outside of current web browsers. This resulted in the development of E15, a three-dimensional Web visualization platform. As a development tool, E15 allows programmers to experiment and develop new ways to display and interact with the Web, without the constraints of a browser.

As a designer creating work on the Web, I continue to find it difficult to design and develop new work for the Web. The visual design and interactions are all dictated by the capabilities of the browser and established web technology standards. We cannot do anything the browser does not enable us to do. Computers are equipped with powerful graphics cards,
and we have an array of new interaction devices, but the browser does not allow access to many of these features. In order to explore new ways to deliver information from the Web, it was essential to develop E15 – and will enable web designers like myself to rapidly prototype new ways to design with web-accessible content.

1.2 Contribution

The goal of the thesis is to develop, understand and evaluate a framework for defining transformations on the Web; and establish a guideline to inspire visual and software design on the Web to leverage their socially influenced, time-based behavior. To meet this goal, the contributions of this thesis includes:

- An overview of the current state of the Web.
- Summary of prior experiments conducted which explore new visual design techniques for presenting information.
- Presentation of research experiments created for the Web environment within the browser.
- Implementation and exploration of a Web visualization platform, enabling visualization of web-accessible content.

1.3 Thesis Structure

Beyond this chapter, the thesis is presented in four additional chapters. In Background, I present the current Web environment as a platform, discuss various approaches to information filtering, and the role of visual systems as an aid to human comprehension. In Experiments, I present various works in two sections: projects developed within the traditional web browser, and design experiments conducted throughout the development of E15. The projects will be presented in detail, and discovery
of concepts supporting the thesis through the development process will be outlined. In Discussion & Analysis, I present an analysis of the thesis experiments in order to identify the particularities of the Web and its socially influenced time-based behavior. In Conclusion, I summarize the motivation and results of the thesis, and discuss future possibilities that have opened up with the work of this thesis.
Background

Roland Barthes proclaimed the death of the author [2] by introducing the idea that writing begins when the indiscernible voices of the texts loses its origin and the author enters his own death. The figure of the author is a production originating from the end of the middle ages. As with primitive societies, narrative is undertaken – not by a person – but a mediator in which the mastery of his narrative code may be admired, but not his genius. Though this idea originates from literary criticism, the philosophy is relevant to other creative fields as well as in scientific and academic fields – including this very thesis. It is not the personal reasons, history nor motivation of the author; but ultimately (and quite literally), the readers alone that determine the success of the thesis and consequently the satisfaction of the degree requirement. In the words of Barthes:

[A] text consists of multiple writings, issuing from several cultures and entering into dialogue with each other, into parody, into contestation; but there is one place where this multiplicity is collected, united, and this place is not the author [...] but the reader.
Marcel Duchamp speaks of the creative act as one which considers both the two poles in creation of art: the artist and the spectator [12]. Similar to Barthes's statement, Duchamp underlines the importance of the spectator (i.e. the posterity) as responsible for contributing to the creative act and deciding the final verdict on its success.

All in all, the creative act is not performed by the artist alone; the spectator brings the work in contact with the external world by deciphering and interpreting its inner qualification and thus adds his contribution to the creative act. This becomes even more obvious when posterity gives a final verdict and sometimes rehabilitates forgotten artists.

To me, these words best describe what the Web has become. In the current web environment, it is the reader or spectator (i.e. the user) that define the success and future evolution of the Web. The user is the one who queries information from a vast array of web accessible information: creating connections amongst the disconnected and contextualizing information in order to obtain what they are looking for. The work of individual web content creators, designers and developers are only as meaningful and successful as measured by the user.

In this chapter I present the relevant background material in three sections. Section 2.1 describes the current state of the web as a platform. Section 2.2 describes various methods of information filtering. Finally, Section 2.3 describes current research and examples of work from multiple disciplines.
2.1 The Web Platform

The World Wide Web began with a simple goal: to allow “high energy physicists to share data, news, and documentation.” Proposed by Tim Berners-Lee, at CERN in 1989 [6], the initial release contained a web server, and a web browser which were both developed [5] using NeXTStep in Objective-C. This project with a simple goal has become one of the most important technological innovations in the last century. Not to discredit the work of Berners-Lee, dramatic evolution and transformations occurred through users of the original web; and the initial motivation as a tool for physicists is all but irrelevant, except for sake of historical context.

2.1.1 Web 2.0

First coined by Tim O'Reilly, Web 2.0 [28] is not a technological improvement over the previous generation of the web. Instead, Web 2.0 is a post dot-com bubble realization that the Web is a platform, and changed the way software developers and end-users interact with the Web. Web 2.0 made the Web more usable and social. It created an opportunity for the Web to not only become a source for information broadcast and consumption; but enabled end-users to dynamically send and receive data, which produced many useful web applications. With many of these web applications, user participation and contribution plays an essential role. The success for these applications can be attributed to the fact that many people are now connected online – all the time. They can treat remote applications as if they were local, and web applications also have the added advantage of location independence. Users are not bound to

---

1 It is a nice coincidence that E15 being developed almost twenty years after the first browser is also developed using Cocoa (which is based on NeXTStep) in Objective-C.

2 Core browser functionality like XMLHttpRequest or data passing between hidden <iframe> techniques have already existed during the Web 1.0 era.
use the application on a particular computer, they can access their data with any computer as the whole application lives on a remote server. All that is required from the user is a compatible web browser.

Beyond new applications exclusive to the Web, there is a trend to turn traditional desktop applications into pure web applications running inside a web browser. This paradigm provides benefits for both software engineers and end-users. Since the application is accessed through a web browser, software engineers only need to develop for a single server platform on the server end – effectively making the application platform independent with the exception of browser compatibilities issues on the front end. End-users get the benefit of always working with the newest version of the software, never having to install anything and ensuring compatibility by using a supported web browser.

This also changed the notion of the software release cycle. New web applications were launched incomplete, as beta; where often applications will have features that are undocumented or unsupported but could be tried out by users. Referred to as “the perpetual beta” [28] – Web 2.0 created a culture in which software (often not rigorously tested or evaluated) are made available to users without ever leaving the beta stage, forgoing the traditional software development timeline. Feedback to the development team is often encouraged, thereby inviting users to take an active role in the development process. Many developers work on their software openly, frequently engaging with users, thus reiterating the importance of the user.

Google for example has numerous “office” applications bundled as Google Docs; which includes a word processor, spreadsheet and presentation software. All files are stored remotely, hence with access to a supported web browser and an internet connection, you will have access to view and edit all your files. As broadband connectivity has become ubiquitous, the usability of these applications increased dramatically. This environment
produced consumer products like gOS. gOS is a Linux distribution based on Ubuntu, specifically optimized for use on low end computers. The optimization is done through various aspects of minimizing hardware requirements, such as: replacing the window manager from GNOME/KDE with Enlightenment to lower CPU/GPU requirements and replacing desktop applications with web applications where possible. Since web applications store files onto remote servers, relying on web applications will enable gOS to manage with a small hard disk. With the minimal requirements of gOS, gPC was developed as a low cost linux PC, distributed through a major retailer. In essence the gPC is a thin-client, and depending on its success, a low cost thin-client can become a viable PC.

The web as a platform, with its ever growing number of available applications; highlight the importance in the role of users. In the case of software development, the programmer takes on the role of the author, the user with the role of the reader. The essential usefulness of the software is not drawn from the individual programmer, but from the collection of its users. What is Facebook without its daily users, or YouTube without its content creators? Users are not only users of the software, but are participants whose daily activities within these communities maintain their existence.

2.1.2 Networked Environment

With millions of personal computers permanently connected to the Internet, this networked environment has become a successful application domain through distributed computing. Projects such as SETI@Home and Folding@Home use CPU cycles of contributors’ computers when they are idle for processing small data-sets. Once a data-set is processed, the result is submitted to the project server and a new data-set to be analyzed is downloaded. With this distributed computation model of sharing idle CPU cycles from many computers, complicated computation can
be completed without the need of an expensive, singular supercomputer. The DIMES Project is based less on actual CPU cycles of contributors, but takes advantage of contributors' physical location and their network connection. The project attempts to create a map of the Internet [Fig. 2-1] from the bottom-up by installing a small daemon application that run periodic pings and traceroutes from contributors' computers of which physical locations are known. These projects take advantage of contributors' good will, without the need for any prior knowledge or skill.

2.1.3 Social Problem Solving

Beyond connected computers, the idea of connectivity between people is powerful; as defined by Human Computation [38] and Peer Production Systems [4]. Human computation and peer production systems use humans to solve problems that are hard computationally but are easy for humans. For instance, recognizing numerous objects in a photograph, and creating a meaningful semantic label is a hard computer vision problem which can easily be solved by a human. Luis von Ahn's ESP Game [40] works in this fashion. A photograph is shown to two anonymous players. Each player uses words not included on the taboo list to describe the
2.1 The Web Platform

photograph, in an attempt to match the word with your partner [Fig. 2-2a]. The side effect of the game is that you obtain an accurate semantic set of words that labels the photograph. Google Image Labeler [16] is a commercial application to the ESP Game that improves the relevance of images searched through google. NASA's Clickworkers [24] is a project where non-scientists can contribute to scientific research [Fig. 2-2b]. Given an image of a small Martian surface, a Clickworker identifies all the craters they can find within the image. The volunteers need very little scientific training and instead uses their human perception and common sense to achieve the task.

![ESP Game in play.](image1)
![Locating craters in Clickworkers.](image2)

Fig. 2-2 Examples of web applications in human computation.

There is high potential for using humans for complex tasks, and with this in mind Amazon created Mechanical Turk; a marketplace for outsourcing intelligent tasks. Workers are paid to complete a certain task (called Human Intelligence Task, HIT) which is submitted by businesses who decide that it is more economical for humans to complete the task than to develop a software solution. For software developers, Mechanical Turk provides a web service API which may be directly integrated into their code as a remote procedure call; thereby replacing the brain of the software with a human. If the ability for a human to solve a specific task can be matched by a software program, then as von Ahn argues [39] that is a solution to a hard AI problem.
Most of the HITS available to workers with no previous experience is simple. They usually require human skill with no specialized training. Personally, I spent some time working on a few tasks. While most of the tasks initially sounded easy, to complete them took quite a bit of effort. For instance, one type of HIT required me to obtain contact information (telephone number, email address) for a particular business, given only the name and mailing address. These tasks usually require a little bit of common sense to filter through some web search results. I did a few of these jobs which earned about a nickel each. Another task was to transcribe a short instructional video. This was a much more straightforward but time-consuming task. This required some time to get used to at first, but it did not require much technical skill. It took about a half hour to transcribe a two-minute video, with a whopping $1.00 reward. For my career as an Amazon HIT worker, I spent about three hours on two separate occasions to earn $2.41.

Most HITS depend specifically on human intelligence skills. However, humans also have creative and artistic skills that are also complex computer science problems\(^3\). Aaron Koblin posted artistic tasks to Mechanical Turk in order to create The Sheep Market. With the wage of $0.02 per drawing, the workers were asked to *draw a sheep facing left*. In a period of 40 days, the project ended with a collection of 10,000 sheep, each of which is viewable on the project’s website.

Social collaborations can extend beyond problem solving – to discovery or investigations of data through online discussions. Heer et al. [19] created *sense.us*, a website enabling asynchronous collaboration to construct variety of visualizations. Participants of the site can view, share, discuss and graphically annotate various data visualizations. The Web, with its social environment is an ideal candidate for combining visualization to social collaboration. Deeper understanding or *sensemaking* of visualized data is often enhanced through discussions between groups of people; through

\(^3\)Such example is Non-Photorealistic Rendering which will be mentioned in §2.3.3
2.1 The Web Platform

Fig. 2-3 The Sheep Market created artistic tasks for workers in Mechanical Turk.

their agreements and disagreements on how they interpret data individually. sense.us is a model of asynchronous collaboration system. This is unique with respect to traditional collaboration systems which provide synchronous or collocated models.

Asynchronous collaboration provide a degree of independence [13] amongst each collaborator. Because of time independence, asynchronous collaboration has the convenience that we enjoy with asynchronous communication such as email. Users have the advantage of working when they want, where they want. This presents a unique challenge to developers of such collaborative system – but with the Web being inherently asynchronous, many of the challenges that exists in developing and using software asynchronously has been learned.

The sense.us visualization system [Fig. 2-4] provides a suite of visualizations displaying United States census data from the last 150 years. The system provides users with four main features in order to facilitate collaboration which deepens the understanding of various census data. The
Fig. 2-4 sense.us collaborative visualization system, main interface components: (a) interactive visualization applet. (b) graphical annotation toolbar. (c) bookmark trail of saved views. (d) comments field. (e) comments for current view. (f) url of current state of the view.

features include: doubly-linked discussions, graphical annotations, saved bookmark trails and social navigation. The ability to create graphical annotations is particularly interesting. The annotations can be used as both a pointing device or used as playful commentary. Users may create annotations using standard tools, such as: Text, Ovals, Pencil, Lines and Rectangles – all tools familiar to most computer users. By graphically marking regions of interest, annotations highlight areas in which users can reference in addition to their comments or gesture through simple drawings. For example, Fig. 2-5 shows a graphic annotation used in a visualization of stock market data which includes the period of the great depression.

2.1.4 User Identity

We have talked about how the Web is a social platform, and it is natural to talk about user identity. When I say user identity, I do not mean
personal identity or digital portrait of people portrayed on the Web (or other virtual environments like Second Life). User identity is as simple as authentication credentials – a login name and a password for a particular website. Most websites require a user to authenticate with a unique login and password in order to participate and access the site. With increasing number of websites that require authentication, we have problems keeping track of all the login information as each website implements their own separate authentication credentials.

Major companies such as Google, Yahoo!, Microsoft, AOL and Facebook have all introduced a single login for all websites they operate. They also provide APIs for other websites which can integrate their authentication service – however, none have emerged as the standard. There is an emerging standard, which is open source and free, called OpenID. OpenID provides single login for all websites that support it. It is an open, decentralized, free framework for user-centric digital identity. Companies like AOL, Microsoft, Sun and Novell have already commited to supporting OpenID, and additional support is growing rapidly. Much like current websites, OpenID users have a unique login name and password. The login is a URI, which guarantees uniqueness throughout all sites.

The web experiments outlined in the next chapter all support OpenID and the projects OpenLocker and FakeID (§3.1.5) specifically deals with OpenID and user identity issues.

### 2.2 Information Filtering

The world is full of information. Information surrounds us in physical forms (eg. print, storage media) and channel flows (eg. TV, Internet). According to Lyman and Varian [22], the world produced 5 exabytes of

---

4 This also raises a major security issue as well.
new stored information and an additional 18 exabytes of streaming information in 2002. In terms of digital information, the study offers the following:

- The World Wide Web contains 170 terabytes of information on its surface.
- Instant messaging generates five billion messages a day.
- Email generates about 400,000 terabytes of new information each year worldwide.

With the estimate of new stored information growing at over 30% a year, the task of obtaining wanted information becomes a greater challenge.

### 2.2.1 Data Types

In response to the need for a new kind of filing system for the “memex,” Vannevar Bush [8] stressed the ability for each item in a collection of materials to be linked together in multiple ways. The idea of associative trails was later coined hypertext by Ted Nelson [25] of which along with Douglas Engelbart are credited with the invention through their respective projects: Hypertext Editing System (HES, 1968) and oNLine System (NLS, 1968). Although the web is known as a hypertext system, it is not only a collection of associated trails of text, but a rich collection of intertwined media including text, images, videos, audio and other digital files. This non-linear medium of information was defined by Nelson as hypermedia [25] and represents the data types of the current Web. Although the Web continues to deliver information in new ways, it is still primarily a collection of linked pages, where each page is composed of text, links, images and other multimedia content.
2.2 Information Filtering

2.2.2 Search Engine

The most common way for seeking information on the web is through the use of search engines. Users search for pages of interest by supplying the search engine a search query. A search query can be as simple as a series of keywords to something more advanced by supplying additional information. Some services, such as Google provide operators to construct advanced queries [15]. The search query is evaluated [7] and through filtering techniques such as Page Rank [29], web pages are returned in order of relevance by comparing against a database of indexed sites. Search engines are fully automated – returning relevant results based on search queries is a hard problem. This is a fundamental research topic, especially in a volatile, information-rich environment like the Web.

2.2.3 Semantic Web

The semantic web is another approach to identify individual web content. Envisioned by Tim Berners-Lee, the semantic web is an evolution of the Web such that it becomes capable of analyzing the content, links and transactions between people and computers. With HTML, pages can be categorized by providing metadata within the <meta> tag with a name-content (i.e. key-value) pair – however, it cannot describe arbitrary things. The Semantic Web employs the Resource Description Framework (RDF) [17] as a way to model information. By using subject-predicate-object expressions (ie. triples) RDF can describe a resource (subject) with traits (predicate) between the resource and the object. For example, using RDF we can define a product for sale on an e-commerce site that: “product 1234-01 is a book on sale with price of $39.95,” by specifying “1234-01” as the resource, “is a book on sale with price of” as the predicate and “$39.95” as the object.
2.2.4 **Software**

When we interact with physical objects, we have the ability to transform their properties. Some transformations are reversible, while others are not—the objects can either be augmented or destroyed. These transformations add some kind of story or history to the objects. In computation, physical input devices go through this transformation. For instance, well-used keyboards and mice begin to show wear, which can reveal certain information about the owner of the devices. Keys that are used frequently are generally shiny through wear, while others begin to accumulate dust, and depending on which side is more worn, a mouse can reveal the owner's handedness. These subtle clues directly or indirectly, consciously or unconsciously provide additional information.

Unlike physical objects, software does not have inherit properties that enable transformation. An application that is used more often does not show wear over time, or a file not frequently used is not covered in dust. Using metaphors of physical transformation can become a useful indicator. Wear-based filtering is another method of information filtering, based on usage. Wear-based filtering can be useful for individuals, but it is especially useful in collaboration settings.

Hill et al created Edit Wear and Read Wear [21]; a system in which histories of a document’s editing and reading are displayed to the user. The histories are displayed on the scrollbar, through the use of *attribute-mapped scroll bars* [Fig. 2-6]. Edit Wear and Read Wear both graphically display the location of wear, the lines in which editing has occurred or have been read. This tells the user what sections of the document are most popular or how often do certain sections get edited. By taking the normally private activity of reading and editing documents to a public one, collaborators can benefit by identifying sections of activity easily. Each collaborator belongs to certain categories, such as (Mary Doe, research, unix expert) in order to label wear and serve as indices in the history.
2.2 Information Filtering

Fig. 2-6 Attribute-mapped scroll bars in various modes: (a) is a normal scroll bar without wear; (b) snapshot of edit wear; (c) snapshot at a later stage with two categories of wear displayed; (d) showing total read wear and (e) read wear showing various categories. [21]

DeLine et al. [11] used the idea of computational wear into a team based software development environment [Fig. 2-7]. Long term collaborative software development is a complex task. It requires programmers to navigate through source code containing large number of files. Documentation is important in this environment, however it is often missing or out-dated as it is expensive to maintain and produce. By evaluating problems faced by current programmers, DeLine et al proposes a system where each programmer’s activities are recorded and visually displayed to address the problems of: 1) the need to scan most of the source code to identify important pieces and 2) getting lost while exploring the source code.
2. Background

Fig. 2-7 UML of source code, showing heat map of wear [11].

2.2.5 Social Filtering

The Web is a social place, and information can be filtered by individuals and communicated to other users. Since interest is subjective, it is a hard problem for any computer program to label what information is interesting, but it is easy for a human to do so. Social filtering comes in various forms, and many sites employ this filtering system. For instance, Alexa collects and provides web traffic information of a website to other websites. By collecting navigation habits of its user community, Alexa

Alexa Internet
http://www.alexa.com

Slashdot
http://slashdot.org
provides users recommendations on where to go next. Sites like Slashdot constantly provides links to a particular genre of information (e.g. news for nerds) submitted by readers and posted by moderators. Social bookmarking services such as del.icio.us keeps track of how many people bookmark certain sites, and can keep you informed of which sites are popular. Digg displays links to pages that have been marked as popular by its users, and web developer can integrate a web badge on their pages so that readers of the page can increase the count by clicking on it. Facebook allows users to submit links to share amongst their friends. There is also a Facebook application that allow users to publish their del.icio.us bookmarks.

2.3 Visual Systems

Visual systems acts as external aids to facilitate our understanding of what we see and what we think. We rely on development of these external aids to make us smarter. According to Tufte, abstract pictures to show numbers were not invented until 1750-1800, long after the triumphs of mathematical ingenuity [37]. These data graphics are not just substitutes for replacing numbers in statistical tables, but they become instruments for understanding information. Just like the continued development of telescopes which enable us to see further out into the universe or particle accelerators that enable us to “see” elementary components that make up the universe, evolution in visual systems enable greater understanding of information. We are limited by our physical capacities – evolutionary improvements are bound by the slow evolutionary process within our biology. So we depend on technology and continued development of instruments to expand our limitations.
2. Background

2.3.1 Human Capacity

In order to design effective visual systems, we need to understand how humans process visual information. The Model Human Processor (Card, Moran and Newell [9]) is an abstract engineering model of human information processing [Fig. 2-8]. The flow of information processing can be described as a system composed of the following abstract components:

- **Short-term sensory store** similar to the frame buffer in computer graphics, sensory input from the eyes and ears are stored with a decay of about 200 ms for visual information.

- **Perceptual processor** recognizes and extracts symbols (with help from long term memory) from the stored sensory input.

- **Cognitive processor** makes decisions based on the recognized symbols, in charge of all the “thinking” that is based on skills, rules and knowledge.

- **Motor processor** takes decisions from the cognitive processor and instructs the muscles to execute. This process may provide feedback loop to the senses.

Crapo et al [10] outlines important visual features that humans recognize. Particularly, the features of motion, color, intensity, size, intersection, closure, orientation, lighting and distance can be recognized by the perceptual processor – within 100-200ms. We can detect changes without the cognitive processor, or without thinking. By using these visual features in software, we can provide users to rapidly recognize areas of interest, we require the use of these visual features. The timeframe implies that we cannot detect change of less than 100ms (perceptual fusion). Therefore, visual systems cannot change states or create changes in information within that timeframe. This also suggests that if we want to convey smooth animation, we need to present frames greater than 10 frames per second.

Another important concept for information processing is chunking. A “chunk” is defined as a unit of perception or memory. Symbols that we recognize are chunks and we have the ability to form and learn new chunks.
For example, common three letter acronyms like IBM, ABC and CIA can be chunked as a group to convey meaning. On the other hand, with random letter combinations, each letter is considered a group. With careful consideration, we can represent data in chunks such that we can minimize the amount of data required to present information.

Humans can focus their attention on only one input channel at a given time. Therefore, a *spotlight* metaphor is used to describe this effect. The spotlight moves serially, from one input channel to another (i.e. visual to auditory). Once attention is focused on a particular channel, all stimuli within the focus area are unintentionally processed. This causes interference, which can be demonstrated by the Stroop effect. The Stroop effect can demonstrate that interference can occur between semantic meaning.
of a word and what we see; which leads to delay in reaction time and increase in mistakes. To illustrate this effect, John Ridley Stroop conducted two tests, one where participants repeat the written meaning of words with differing colored fonts (RCN – Reading Color Names), and the other where participants orally identify the color of each printed color name (NCW - Naming Colored Words) [Fig. 2-9]. Even through practice, participants still took an increased time to complete the NCW task due to interference. With automation of reading, the mind automatically determines the word’s semantic meaning. The color identification on the other hand is not automized, and the task of overriding the initial meaning of the word is thought to be the cause of the interference. We should therefore be careful when choosing secondary characteristics of visual systems. The stimulus should reinforce the message, not interfere.

Fig. 2-9 Stroop Effect: Ignore what each word says and name the color each word is written in. Compare how much longer it takes for the right column compared to the left.

In the winter months, I run in the 200-meter indoor track at MIT’s Zesiger Sports and Fitness Center. The track, like many of the other facilities at the gym is multipurpose. The floor has many markings for different events, in various colors – providing contrast from one function to another. From the spectator point of view, the contrasting lines may be confusing or irritating. However, as the runner, in the moment when all
your focus and energy goes to concentrate on the task of running, the visual contrast works in such way that you can effortlessly see the lines in which you have to follow.

### 2.3.2 Aesthetics and Graphic Design

It may be superficial to talk about what something looks like, but as a graphic designer I acknowledge the importance of aesthetics and how graphic design is used to communicate information effectively. Whether something looks good or communicates well is subjective and often cultural. However, we can analyze aesthetics in terms of artistic and design goals. Additionally, we can certainly develop objective and quantifiable qualities.

Graphic design is as old as the history of mankind. Cave paintings in Lascaux has presented visual communication since 14,000 BC [Fig. 2-10]. The goal of graphic design is to visually communicate information using images and typography. Various techniques in graphic design has been developed over the centuries, and these techniques for effective visual communication can be borrowed to improve the conveying of information using graphics and typography.

On the Web, graphic design exists as web design, where mixture of images and typography exists within individual webpages. Even though web design is constructed through markup and technologies specified by standardization – like all fields in design – there is no standardization of design on the Web. Each designer can present information in any way they want, limited only by the web technologies that exists. With this freedom of artistic expression also comes with a price. Because there is no standard way in which information is presented, each webpage is designed without a standard method to extract specific information. For instance, suppose we wanted to check the weather in a small town in the US and
a small town in Canada. In order to obtain the weather forecast of the small town in the US and Canada, we have to visit two separate sites, http://weather.com and http://weather.ca, respectively. Each site is completely different – the markup as well as the visual design – and there is no consistent way in which we can query the forecast the same way.

There are established guidelines on how best to design webpages to maximize usability, such as Jakob Nielsen’s “Ten Usability Heuristics” [26]. However, the web experience through the web browser is mostly read-only; therefore we cannot modify information or the design of a page. For the most part, we have to deal with consuming information from badly structured webpages.

Web design has been indirectly standardized through the widespread use of web publishing software. There are numerous open source blogs and wiki systems, and no one needs to write their own software from scratch. Blog systems like Wordpress have a template-based design system which are independent of content. Users can easily change the design of the site by simply changing which template to use. There are many freely avail-
2.3 Visual Systems

able templates and advanced users can also create their own. Many blogs and wikis look similar because many users do not bother or know how to change the template, and by establishing a template which adheres to usability heuristics, we can establish a more usable Web. This of course comes with a design backlash – websites begin to look the same and design variations on the Web will decrease.

2.3.3 Non-Photorealistic Rendering

The goal of computer graphics began with a goal to generate an image that would achieve realism indistinguishable from photography. With over 40 years of history, computer graphics today can achieve this goal. Hollywood films for the last decade has created some stunning realistic graphics that can make anyone not aware of the technology to believe as real. With the initial goal accomplished, there is growing interest in producing graphics that convey information to humans. Conveying information with a photorealistic image is a challenging task. You can see that from history of art, photorealistic paintings never existed until the works of American photorealism painters of the 1960s and 70s such as Richard Estes, Ralph Goings and Chuck Close [Fig. 2-11]. Throughout human history, from the drawings of ancient Egypt to medical illustrations in current textbooks; drawings were created without the constraints of reality. Drawings were not photorealistic, they depicted an image that often distort reality to create a point of view. This was not because it took the technical skills of the photorealists to finally create photorealistic renderings with human hands, but rather, there are human tendencies to effectively communicate information. When we sketch out ideas, we draw things that represent an idea, not necessarily what something looks like in reality.

The field of Non-Photorealistic Rendering (NPR) created a new discipline of computer graphics beyond photorealism. The most relevant goal of
NPR in the thesis is a goal to create renderings of images that can convey meaning. This inherits an artificial intelligence problem of how to create renderings that will convey meaning within the image. One way in which humans convey information is by constructing caricatures. Caricatures exaggerate a certain essence of a person that makes them easily identifiable. Often, they emphasize characteristics that are not flattering, and used widely in humor and propaganda.

### 2.3.4 Three-dimensional Information Landscape

Works of information landscape, a space in which information hangs in a three-dimensional environment where users can freely fly from place to place; has been explored and designed by Muriel Cooper and her students in the Visible Language Workshop (VLW) at the MIT Media Lab. They argue that the use of three-dimensional presentation is effective in visualizing large, complex information spaces. Although there exists a rich history of typography in traditional two-dimensional graphic design, three-dimensional typography – the design and technical issues surrounding it, have only recently been explored. The Visual Language Workshop have contributed extensive work in this field through their experimental software tool designed to investigate the use of interactive three-dimensional
2.3 Visual Systems

environment as a medium for typographic communication.

With Typographic Space [34], the VLW was able to lay out text in a large three-dimensional space [Fig. 2-12], change basic sets of typographic attributes while using a simple interface consisting of a keyboard and mouse for the user to fly through the space. The keyboard controlled the rotation and translation of the viewport, while the mouse changed the view distance. They identify issues regarding three-dimensional typography, including:

Distortion of typographic form from various viewport perspectives.
Loss of meaning in apparent type size.
Loss of type size as a cue for the perception of depth of field.
Identifying three-dimensional motion as a new expression in typography.

David Small's Virtual Shakespeare Project [33] illustrates solutions to some of these issues. The project is a design experiment where a large body of text is displayed to a screen through the use of variable-scale typography, visual filtering techniques and organization of complex relationships among different elements of information. The textual information visualized in the project is the complete plays of William Shakespeare, which places the amount of text on the order of one million words. The work itself also contains many structures such as: speeches, scenes and acts – which can be made visible for the visualization. Moving along distances – or analogously through scale, various amounts of words are displayed. If the viewport is close to a text, it is fully rendered. However, as the viewport moves away from the text, simplified textures [Fig. 2-13] in place of each line of text is rendered to illustrate the structure of the text. This technique, called greeking shows the underlying structure of the text better than attempting to render text that is too small and illegible. The structural information, though continually more abstract, informs users of the texts’ location in reference to the larger environment, and its relative scale compared to other elements.
2. Background

Fig. 2-12 Example of three-dimensional space occupied by large amount of text. Shown here, David Small's Virtual Shakespeare Project.

2.3.5 Visual Web Systems

Since the early days of the Web, visual explorations of the web have been explored in an attempt to extend our interaction capabilities currently limited by browsers. Data Mountain [32] is a three-dimensional document organization system that is capable of organizing web pages. The 3D interface [Fig. 2-14] was designed specifically to take advantage of human spatial cognition. When users are given the ability to place documents freely, they impose their own rules on where certain documents are placed. A 3D desktop environment is used instead of 2D in order to display more information without incurring additional cognitive load due to pre-attentive processing of perspective views – the differences in sizes
2.3 Visual Systems

Fig. 2-13 David Small's Virtual Shakespeare. Four views of the complete collection of William Shakespeare at different scales.

indicate spatial distances. The software effectively functions similarly to bookmarks in browsers; and the authors claim that compared to the Microsoft Internet Explorer Favorites (i.e. bookmark) mechanism, Data Mountain has reliable advantages.

Pad++ allows the exploration in visualizations of graphical data with a zooming interface. Much like E15, Pad++ is a general-purpose substrate, and not an application on its own. Using a Tcl/Tk interface, multi-scale graphical objects can be created and manipulated while navigating through the object space. Using a zoomable user interface (i.e. ZUI) Pad++ enables abstract object representations with semantic zooming. Semantic zooming is similar to the natural mapping of viewing information. De-
2. Background

Fig. 2-14 DataMountain showing 100 webpages in a 3D desktop environment. Users are allowed to organize webpages freely within the interface.

tails of an object are shown when zoomed in and viewing the object up close. However, when zoomed out, rather than displaying a scaled down version of the same object, a different representation of the object can be shown; similarly to the effect of greeking with text. This effect is potentially more effective in describing the object at the coarser level.

With many hypertext systems, graphic depiction of important semantic relationships do not exist. Typically, a new window is opened or window content is replaced with new content when users click on a hyperlink. This behavior does not represent the important parent-child relationship that may exist – just as with most window systems, the windows are separated and do not provide any graphical representations of relationships between the windows. One such solution is to use an inline approach, where hypertext information is presented within the original document. An example of this is the built-in dictionary lookup in Apple's instances of NSTextView.

Fig. 2-15 Example of inline hypertext: dictionary in Mac OS X.
2.3 Visual Systems

Using Pad++, a hypertext interface has been explored by Benderson and Hollan [3] through PadPrint [20]. With the multi-scale layout capabilities of Pad++, parent-child relationships are graphically represented between links. When a link is clicked, the linked data (i.e. web page) is scaled down and loaded to the right of the current document [Fig. 2-16]. Also, the view is animated to center around the new content. Shift-clicking any of the objects will load its parent view, thereby retaining the parent-child relationship of the documents.

Fig. 2-16 Hypertext: Pad++ for displaying graphical representations of hypertext.
3

Experiments

The goal of the thesis is to develop, understand and evaluate a framework of ideas in designing webpages and web data with their socially influenced, time-based behavior. There are a variety of experiments performed during my two years of research at the Media Lab. Some projects are more relevant than others, and this chapter details these experiments that help support the thesis, presented in two sections. In Web Applications, I present projects that represent web experiments within the current browser paradigm. In Visual Web System, I present E15, various implementation details and its applications. By extending the Web interaction beyond traditional browsers, we create a new web visualization system.

3.1 Web Applications

As we have seen from the previous chapter, the web is no longer simply a collection of hyperlinked pages of static content. Many websites are sophisticated web-based applications, which may be novel, useful or
even essential. In this section, I will present numerous projects I have designed and developed which are explored through interactions with the web browser. We begin with PLWire, the group webpage for the Physical Language Workshop, where information about the group is broadcasted without the use of many words, and content is changed regularly. Open-Code, a creative tool for graphics programming, which explores the ideas of graphics code writing, editing, compiling and sharing. GPC is a graphical pen-based CAPTCHA system which uses the more physical gesture of tracing abstract shapes as a CAPTCHA test. MudSketch explores the ability for drawing natively within the browser. OpenLocker and FakeID are both anonymous OpenID authentication services. OpenLocker provides an alternative physical metaphor of a locker as a replacement for the traditional user-password combination; FakeID explores the use of using fake information online, while visually exploring how to intentionally make a web page look “dirty.” Finally, RunLog and RunLogger explore how a simple daily logging routine can be enriched socially on the Web.

3.1.1 PLWire

Soon after joining the Physical Language Workshop in the summer of 2006, I began working on PLWire [Fig. 3-1]. PLWire is the group webpage for the Physical Language Workshop, broadcasting information about the group and our current projects. The information is posted without the use of words; instead, we establish four data types: Videos (Cyan), Graphics (Pink), Links (Green) and People (Brown, Orange, Yellow, Black). Each element belongs to one of these data types, and they are represented on the site as a color-coded, draggable element. Each person is also subdivided into four categories, which is reflected by their color: Faculty/Admin, Grad Student, UROP and graduated students.

When the user first enters the site, all elements drop in randomly onto the screen. The user can sift through the elements, moving them around
3.1 Web Applications

Fig. 3-1 PLWire, group webpage for the Physical Language Workshop.

one by one. They can also close the element, by clicking on the close button. The elements are not deleted, but they disappear and will only reappear by reloading the page. Each element also carries a name, which is clickable. The behavior of clicking on the name is dependent on the data type. Clicking on a video will load and play the video on top of the elements, and similarly, clicking on a graphic will display the full size view of the image. Clicking on a person will take the user to their own webpage, while clicking on a link will take the user to the link destination.

To keep the site simple, there are only two additional buttons. One links to the MIT Media Lab website, while the other is an organize button that will take all the elements on the screen and automatically moves it to one of the four corners, organized by data type.
3. Experiments

Implementation

The contents of PLWire is controlled by members of the PLW. There is an administrative interface to the site which lets any of the members create, edit and destroy elements on the site. It is not necessarily curated, instead each member edits the site as they please, creating its own interesting behaviors within the group. Since the site is constantly changing, we keep a daily archive of the site through a cron job that runs once a day. The archives can be accessed using the following hidden url of the form:

http://plw.media.mit.edu/history/YYYY-MM-DD.html

For example, the site on January 1, 2007 can be accessed at the address: http://plw.media.mit.edu/history/2007-01-01.html.

PLWire is a Ruby on Rails application. Specific details of how we currently run web applications using Ruby on Rails is summarized in the end of this section (§3.1.7).

Results

PLWire has been live since the summer of 2006. The site has undergone many content changes over the last two years. But aesthetically, the site looks and behaves the same. One of the more interesting aspects of PLWire is that content change is anonymous. There is only one login to the administrative interface, so there is no way to determine who made the change. Because of this, there are opportunities for a member to post something only to have it removed by someone else. All we can do is speculate amongst one another, creating an interesting tension for what is broadcast worthy for the PLW.
3.1 Web Applications

3.1.2 OpenCode: Social Creativity and Education

In collaboration with Kyle Buza, we created OpenCode. OpenCode is a web-based graphics programming environment, enabling novice and expert programmers to write, edit, compile and share graphics programs—all within the OpenCode website. Graphics programming (or programming in general) are often confusing and complicated for beginners, and the barrier of entry is high. Just displaying a window with a draw-able graphics context can require quite a bit of code. Simplifying graphics programming has been explored in the past, such as Design by Numbers [23] by John Maeda, and Processing [31] by Casey Reas and Ben Fry. Although these systems have lowered the barrier of entry and eased the process for creating graphical programs, we can improve this system further by including the social network component of the Web.

We learn programming through experience and examples. Although important learning occurs through reading references and textbooks, to become a proficient and practical programmer, we need experience through the practice of writing real code. Learning by example and learning from other programmers' code is an extremely helpful way of obtaining required programming knowledge as well as inspiring new creative ideas. OpenCode creates this environment of contributing and consuming code, in addition to the facilities for compiling code directly within the site itself.

The user logs into the site by creating an OpenCode account or through their OpenID. Once logged in, the user can search through a vast collection of user contributed code, and load the code into the text editor by simply clicking on it. The code loads and the user is free to modify the code [Fig. 3-2]. When the user wants to compile the code, they simply click the "Run" button [Fig. 3-3]. This creates an asynchronous HTTP request to the web server and posts the code to the compiler which resides as a web service on the web server. The compiler will compile the code
as a Java applet and sends the applet back to the browser and presents it
to the user. If any compilation error occurs, an error message is sent and
notifies the user of the actual compile error.

Fig. 3-2 OpenCode interface for the code editing page, showing the editor on the left and
code repository on the right.

Implementation

A couple novel ideas are implemented to maintain the community interac-
tion. First, users are free to use the site for their own use. They can keep
all the source code for their programs hidden from public view. By keep-
ing their source private, their applet will also be hidden and inaccessible
to anyone but the original programmer. In the spirit of open source and
to enable others to learn, any program that programmers want to show
publicly will also have to disclose their source code. Thus, for OpenCode
users, any program that they see on the site, they will be able to see the
3.1 Web Applications

Fig. 3-3 Running an OpenCode program within the browser window.

full source and find out how that program was written. Secondly, all code on the OpenCode site are guaranteed to compile or "work." Every time a programmer saves their code, the code is submitted and compiled on the server. If the code is compiled successfully, the code is saved. Otherwise, the user is notified of the error, and they will have to fix it in order to save. This filters out code that fails to compile. With these two implementations, a self-maintaining community is created; which will grow without the need for moderation by the original developers.

The OpenCode website is developed using Ruby on Rails, and the OpenCode system consists of two servers. A mongrel cluster running the main website using Ruby on Rails through Apache2 serving on port 80, and a Java servlet which exposes the compiler as a web service through Tomcat serving on port 8080. The code written on the editor page is sent to the Java servlet through JavaScript as an asynchronous HTTP request.
3. Experiments

to the Java servlet. The code is compiled and saved as a Java applet on
the server to a web accessible directory, and a response of success or an
error message is sent back as an HTTP response formatted as JSON. If suc-
cessful, a JavaScript function is called which constructs and injects the
required HTML snippet to the JSON. If instead the code does not compile,
the error message produced by the compiler is instead displayed on the
page.

Since the system used two servers operating on two different ports, the
system did not initially work due to the same origin policy built into the
JavaScript interpreter. Every browser implements a set of security mea-
sures for client-side scripting that attempts to prevent malicious code
execution. With the same origin policy, browsers assumes not to trust
any content loaded from another server. Since the OpenCode system re-
quired two servers to communicate, we had to work around this issue. Af-
ter various attempts, the simplest solution was to proxy Tomcat through
Apache2 with mod_proxy. By simply adding a RewriteRule line to the
.htaccess file for OpenCode, we were able to serve all port 8080 content
through port 80.

Results

OpenCode was first released in December 2006. Currently, in May 2008
there are 856 users and 362 contributed code. There were no public an-
nouncements or major advertising attempts. Most of the users came from
word of mouth or referrals from various blog posts that have linked the
site. In April 2008, with Kyle Buza, we presented OpenCode at the AIGA
conference, Massaging Media 2: Graphic Design Education in the Age of
Dynamic Media. OpenCode was well received by educators, as a tool for
graphic design students to explore programming in a classroom environ-
ment. We hope with the exposure to this new community, OpenCode will
gain more interest and activity so we can begin to investigate code shar-
3.1 Web Applications

Ting as an effective means for education and foster collaborative creativity.

### 3.1.3 GPC: Graphical Pen-based CAPTCHA

In order to protect themselves from bots and scripts which construct an automated way to spam or create false accounts, many web service providers implement a CAPTCHA system. The term is an acronym for *Completely Automated Turing Test To Tell Computers and Humans Apart*; coined by Luis von Ahn et. al. [39]. The idea is to attach a validation question to a web form in which an answer would be simple for a human but hard for a computer to solve. Various methods exists, but variations in the original test developed by von Ahn et. al. where users must identify words appearing in a distorted image is most common. Other forms exists, such as audio for the visually disabled; however, most of these systems are English-based and culturally-dependent.

To create an alternative mode of interaction, I developed a graphical, pen-based CAPTCHA system [Fig. 3-4]. The user is given a drawing surface in which series of line strokes are drawn in a precise stroke order to create a pentomino shape. The user must trace over the pentomino shape and stroke the line segments in the correct stroke order. This mode provides an alternative to english word recognition which create difficulty for non-english speakers. Sketching is more culturally independent and can appeal to a wider audience for a worldwide environment like the Web.

**Implementation**

The system is implemented in two parts. The drawing utilities as well as stroke segmentation is performed on the client-side through JavaScript, made possible by `<canvas>`. The segmentation information, where the stroke is divided into individual line segments is sent as an HTTP request.
3. Experiments

Fig. 3-4 Graphical pen-based CAPTCHA system.

to the server, where the recognizer exists. First, the recognizer performs a comparison to the segmented stroke, but as a fallback procedure, a raster recognition is also performed by turning each stroke into bitmap. A success or failure response is sent back to the original page, which will then validate or invalidate the form request.

Results

Initial user evaluation showed that the system worked well with the use of Wacom tablets and tablet PCs. Using the mouse as the tracing input was challenging even after some training time. The raster recognizer was added to improve mouse input results. Integrating alternative but naturally intuitive interaction methods like the pen interface to the Web should be investigated for enhancing the Web experience.
3.1.4 MudSketch: Web Drawing Interface

MudSketch was developed on top of the original drawing work done for GPC. It is a JavaScript library to enable drawing directly within the browser, and does not require the use of Flash or a Java applet. Instead, it uses drawing methods provided by `<canvas>` and is natively supported on all major browsers\(^1\). Sketching, which is a fundamental natural gesture, has been natively possible to implement on browsers only in the last two years. MudSketch supports basic drawing functions such as variable brush size and color. There is also support for unlimited undo/redo.

Although MudSketch was created as a reusable JavaScript drawing library, I built a complimentary web application to demo the drawing capabilities. The complimentary site, also called MudSketch [Fig. 3-5] enables anyone to anonymously draw on the canvas, and save. Any saved drawing can be loaded by anyone, and the drawing can be modified. There are no destructions of previous drawings, since you cannot save in place of another drawing. The site also features an ability to export to PDF and PNG. Any drawing on the site can be exported as PDF or PNG.

Implementation

As mentioned previously, MudSketch was written in JavaScript and supports all major browsers. The MudSketch web application uses JSON as a transport format to exchange drawing data to and from the browser to the web application. Using JSON was simple, since we can instantiate the JSON object as a new drawing instance to load. We just needed to write support for JSON loading and exporting methods. The conversion between MudSketch drawing to PDF was done on the server in Ruby. Using PDF::Writer, a native Ruby PDF library, the drawing data stored as

\(^1\)Support for Internet Explorer is available as a plugin or various JavaScript libraries from Novell and Google.
3. Experiments

Fig. 3-5 MudSketch, a drawing tool written in JavaScript.

JSON is first converted as an ordered array of line segments with brush size and color information. By using PDF::Writer\(^2\), the lines were drawn procedurally in order then finally saved as a file to a URL accessible location. Similarly PNG bitmaps were created using RMagick\(^3\), a Ruby API for ImageMagick\(^4\).

Results

The drawing tool worked quite well, and the web application produced some interesting works. The site currently contains 581 drawings. Most

\(^2\)http://ruby-pdf.rubyforge.org/pdf-writer/
\(^3\)http://rmagick.rubyforge.org/
\(^4\)http://www.imagemagick.org/
of the drawings are just scribbles that people have made in order to test the library, however there were some very detailed drawings created and a couple of them are shown in Fig. 3-6. Also, since the system is anonymous, I do not know who created the drawings. There were many drawings that are typical to a public bathroom wall. Following MudSketch, two projects were produced using the drawing library: *Modster*<sup>5</sup> and *Badger*.

![Fig. 3-6 A couple of detailed images drawn using MudSketch.](image)

### 3.1.5 OpenLocker and FakelD: Web Anonymity

OpenID was introduced in the previous chapter, as a system to provide a single login across many websites (§2.1.4). OpenID is decentralized, and is not owned by any one person or company. Instead, it is a lightweight method to identify individuals. Anyone can choose to become an OpenID user or an OpenID provider for free, without registration or approval from any organization. With this freedom, I developed two projects built on top of OpenID in order to enhance the anonymity of users on the Web.

<sup>5</sup>http://modster.media.mit.edu/
OpenLocker

OpenLocker is an OpenID provider and RSS aggregator. In the specifica-
tion, OpenID requires the login to be a URI, however, it does not specify a
specific authentication method; or in other words, OpenID requires the
username to be a unique web address, but the password can be anything
(or not even have a password at all). Most OpenID providers use an al-
phanumerical password as their authenticating credential, but this can take
on alternative forms such as biometric authentication which include digi-
tal fingerprinting or iris recognition. OpenLocker uses an old metaphor of
locker rooms and combination locks, in order to make the authentication
process easier to understand for all users.

In locker room situations, each person has a locker located in a physical
location (usually accompanied by a number) and kept secure with a sin-
gle dial combination lock. The locker is usually assigned and the combi-
nation on the lock is determined randomly by the manufacturer. All the
security parameters are specified by the locker system, not by the user. It
is up to the user to remember the location and the combination, and it
is not easy to change the locker nor the combination. The users are stuck
with what they get.

OpenLocker provides users with lockers that are assigned to them using
the MIT map [Fig. 3-7]. Each locker location is presented on top of the
MIT map with corresponding building numbers. This translates into a
unique URI. For instance, a locker that exists on top of Building 5 with
the number 123 will carry the following URI: http://lock.media.mit.
edu/locker/5/123.

In order to authenticate using the locker, the user is presented with a sin-
gle dial combination lock [Fig. 3-8a] which they must open. Just like a
physical lock, the user needs to spin the lock twice counter-clockwise in
order to reset after an unsuccessful attempt. When the locker is initially
assigned, the user is given an instruction like the following:

The lock functions as a standard single dial combination lock. Please follow the instructions below to open your locker:

1. Turn the dial clockwise (number will decrease) two full rotations to 10 and stop.
2. Turn it counter-clockwise (number will increase) past 10 to 28 and stop.
3. Turn the dial clockwise to 5 and stop.
4. Open the lock by clicking on the lock, or pressing enter/return.
5. If lock fails to open, restart from step 1.

The single rotary combination lock can be turned by using arrow keys (left-right, up-down) scroll wheels (scrollup-scrolldown) or using alternative peripherals like Griffin Technology's PowerMate. Once opened, the locker acts as a homepage or starting point for the user's daily Web interaction. It contains collections of user configurable RSS modules [Fig. 3-8b]. The user can create new RSS modules by specifying a URL to a valid RSS feed. These feeds can also be easily deleted and recreated. They can
be moved around and positioned much like windows on a desktop. The
position of each feed is remembered so that on subsequent visits to the
locker, the positions are maintained.

(b) Inside a locker.

Fig. 3-8 Opening the locker requires users to unlock a combination lock. Inside the locker, users can create modules of their favorite RSS feeds, which they can move around the locker freely and reposition.

**FakeID**

Like the name implies, FakeID enables the idea of using a fake identity
on the Web with OpenID. Currently, OpenID URIs are unique. The
problem with unique URIs is that like email addresses (which are also unique,) traditional OpenID URIs can easily match digital identities with physical ones. FakeID functions as a proxy to protect the user's real information
and submits fake information to services requesting OpenID credentials.
The user fills in whatever fake identity they want to construct for their
FakeID, and generates a new unique URI.

Once users create a FakeID, they can submit the unique URI associated
with their fake information as their OpenID URI. When a service au-
thenticates against the FakeID URI, the FakeID server will authenticate
3.1 Web Applications

against the user’s real OpenID credentials. Once successful, the fake information will be sent to the original requester. Thus, FakeID is a proxy to the user’s real OpenID.

Implementation

Open Locker and FakeID are built using the ruby-openid\(^6\) library. The ruby-openid library handles the authentication requests and creates the appropriate responses and url redirects to function as an OpenID client and provider. The single dial lock is implemented in JavaScript and all components\(^7\) are drawn using drawing methods provided by `<canvas>`. When the lock is used, the selected numbers are registered one by one and sent as an asynchronous HTTP request for authentication. Once authorized, sessions will be established and the browser will be forwarded to the locker page. The lock page is encrypted with HTTPS and the combination sent for authentication is encrypted also under SSL.

\(^6\)http://openidenabled.com/ruby-openid/
\(^7\)Except the numbers that show up above the lock. Currently, `<canvas>` does not support text rendering.
OpenLocker currently has 74 lockers. People from the MIT community have specifically requested a locker location of their choice (usually the building of their lab or dorm). Since most locker requests come from outside MIT, I have randomly distributed the locker assignments to the Stratton Student Center (W20), Kresge Auditorium (W16), and the J.B. Carr Indoor Tennis Facility (W53). The Media Lab (E15) currently has 15 lockers, all occupied by Media Lab students and faculty.

OpenLocker has been shown to various people since its launch in early 2007. In general, people seem to like the idea of using a locker metaphor for authentication, however it was difficult for them to understand the concept of OpenID. To many people, the idea of a decentralized, foreign authentication service is either confusing or suspicious. However, the thought of having to remember only one authentication credential was welcomed by all, and in terms of OpenLocker, they understood the metaphor and the importance of keeping your password a secret.

The single dial lock as an input device was very familiar to Western users. However, for many users from Asia, the lock seemed to be rather foreign. One particular user told me that he had only seen them in movies that show bank safes. Also, to get used to the lock interface takes some time. However, especially with the use of the PowerMate, once you become familiar, the lock is easy to unlock.

Within the locker, users keep various number of RSS modules. The number of modules is unlimited, and they are able to move the modules around within the locker and the positions are retained. This creates an opportunity for users to have “clean” or “messy” lockers.

Currently, there are 114 FakeIDs. All of the FakeIDs created can be seen publicly on the FakeID website. Initially, the birthdate field had a range
3.1 Web Applications

of 80 years. But I had an anonymous request to lift that restriction, which I did realizing that if this were to be in fact all fake information, there was no need to restrict certain values. I realized soon after that people began to create FakeIDs with identities from popular culture, such as: Optimus Prime$^8$, Richard Feynman$^9$, Che Guevara$^{10}$ and Karl Marx$^{11}$.

FakeID has a certain design aesthetic that was intended to help bring the idea of using a fake ID on the Web. The site was intentionally made to look dirty and worn. Each identity is displayed as a fake government issued ID card, to drive the idea of using a real fake ID. By using the metaphor of fake IDs, we hope to create a better awareness for personal, online privacy as well as the concept of having a universal login system.

3.1.6 RunLog and RunLogger

RunLog and its Facebook counterpart, RunLogger is a social network based on a simple web application that keeps track of how much you run. For frequent runners, RunLog can be a useful tool to keep track of your running goals. But RunLog was built to motivate people who are novice runners. Many people start running, to get in shape as their new year’s resolution. The hard part is keeping up with running regularly. There are existing run logging tools available, but RunLog establishes a social network, where runners can motivate each other. I call this, social network peer pressure.

RunLog has a simple interface [Fig. 3-10]. Logging should be easy and simple as to not burden runners, so the information required by the user is kept minimal. The only required information is to submit a distance for a specific date. Additionally, duration of the run and notes can be

---

8 http://fakeid.media.mit.edu/51287537
9 http://fakeid.media.mit.edu/26481175
10 http://fakeid.media.mit.edu/79759307
11 http://fakeid.media.mit.edu/94543199
3. Experiments

added as an optional field. With the small amount of information collected frequently, various results can be produced such as: overall distance, session average, weekly and monthly totals, etc. The front page is a list of top 10 runners in overall distance and session average, as well as new runners that recently joined. There is also a news feed which any runner can post messages to.

![RunLog Website](image)

(a) RunLog front page, showing top runners, their social profile, log input, running stats and list of favorite people.

(b) Profile page showing calendar, log input, running stats and list of favorite people.

Fig. 3-10 The RunLog Website.

RunLogger is a Facebook application, using RunLog [Fig. 3-11]. The two sites run on the same web application, but RunLogger can be accessed within Facebook, and takes advantage of many Facebook features. RunLogger, by using Facebook uses a popular pre-existing social network which creates a more active social community. RunLogger can also post information on a Facebook user’s profile page [Fig. 3-11b] and news feed [Fig. 3-11c], which not only displays useful information for the runner, but also advertises the running information to their friends through the news feeds.
3.1 Web Applications

(a) RunLogger main page showing top runners in the Facebook network.

(b) RunLogger status appears on individual profile pages.

(c) Each log is broadcasted on Facebook user's news feed.

Fig. 3-11 RunLogger, RunLog as a Facebook application.

Implementation

RunLog is a Ruby on Rails based web application. However, depending on how the web page is requested, the content provided is different. For normal HTTP requests, a RunLog web page is sent as a response, however, for requests that originate from Facebook\(^1\), a FBML (Facebook Markup Language) markup is sent instead of HTML. This behavior is provided through the RFacebook\(^2\) ruby library, which is a port of the official PHP

---

\(^1\)All Facebook applications proxy through Facebook.

\(^2\)http://rfacebook.rubyforge.org/
library distributed by Facebook. This property is convenient as it frees developers from writing redundant server-side code when supporting multiple output formats of the same content.

3. Experiments

Results

RunLog and RunLogger are very popular. Currently, there are 615 runners on RunLog while RunLogger has 7,010 runners. There are over 550 daily active runners using RunLog and RunLogger who have collectively posted over 200,000 run entries. The majority of activity come from Facebook runners. RunLogger was released four months after RunLog was originally released. As soon as RunLogger was released as a Facebook application, the number of runners increased dramatically. The spike was somewhat due to the fact that RunLogger was release two days after Facebook announced their official development platform. This initial surge of interest and lack of competition to RunLogger added to its popularity.

RunLogger did not have to create its own social network unlike RunLog. Instead, piggybacked on the Facebook community, thereby having easier access to a pool of users that are already computer literate and active online. Also, many features were relatively easy to implement given Facebook's development platform, and exposes various locations throughout the Facebook community including individual user profile pages and news feeds.

3.1.7 Implementation Summary

The web applications presented in this section have all been implemented using Ruby on Rails. Ruby on Rails has been the web framework of choice at the PLW in the last couple years as it enables us to quickly develop, de-
3.2 Visual Web System

ploy and scale web applications. It also benefits from a diverse community of Ruby on Rails contributors and user-base who not only contribute useful libraries, but share knowledge and experience for implementing various scenarios; through mailing lists, discussion boards, wikis and IRC chat.

The web applications are deployed on our main production server, a 3.0GHz quad-core Xeon with 16GB of RAM running RedHat Linux Enterprise Version 4. This machine runs the Apache2 web server and MySQL 5 database server. Each application runs multiple instances of mongrel, a light-weight ruby web server on separate ports behind Apache. Apache handles the proxy and load balancing of the mongrel processes. This environment enables us to easily scale each site to meet the request load by adding additional mongrel instances for applications with heavy load. A system diagram for a typical web application is displayed in Fig. 3-12

![System diagram of a typical Ruby on Rails web application with three mongrel processes.](http://e15.media.mit.edu/)

3.2 Visual Web System

E15 is an ongoing group project in the Physical Language Workshop, started in the summer of 2007. To summarize generally, E15 is a graphics platform designed to create dynamic, 3D data visualizations and in-
3. Experiments

E15 consists of an OpenGL-based graphics engine, with an embedded Python interpreter. With the Python scripting interface, programmers write high-level Python code which generate graphical objects and interactions into the main OpenGL-based window. What makes E15 unique from other graphics platforms is that E15 does not have to recompile code, which makes it a highly interactive tool to rapidly prototype graphical elements. As E15 scripts are evaluated through a Python interpreter, programmers can use any of the vast number of Python libraries that are freely available, which makes E15 very powerful and fully featured. An overview of E15 components is given in Fig. 3-13.

3.2.1 E15 Overview

![E15 main interface components](image)

**Fig. 3-13** E15 main interface components: (a) OpenGL-enabled 3D environment. (b) WebKit-based web browser. (c) Error console. (d) Python console.

**Browser Integration**

E15 is intimately integrated with the Web. E15 has a built-in WebKit-based web browser that functions like any other web browser; which can
3.2 Visual Web System

be controlled graphically or procedurally through Python. Programmers can interact directly with the web content that has been loaded onto the web browser and create interactions between the contents of the web browser and E15's graphical environment. This creates an opportunity to visually augment current web pages, as well as interact with data content that exists within the loaded web page.

Traditionally, extracting data from web pages were done using a scraper – where automated scripts will extract data from web pages by parsing the HTML markup directly using pattern matching with regular expressions. The problem with this approach is that it is a brittle method prone to failure. Parsing HTML directly assumes that the markup being scraped is well-formed. Many web pages have malformed HTML that is not valid, such as: missing end tags, misspelled tags, missing quotes, incorrect attribute types, etc.

The best method to extract data would be to access the Document Object Model (DOM) constructed by a web browser from the HTML markup. Once the JSON is constructed, we can freely traverse and extract data from parts of the page we are interested in. The browser will take care of all the markup errors that may exist in the HTML document. Since web developers test webpages based on empirical testing rather than markup validation alone, the way in which a website is rendered through a web browser more accurately resemble the structure of the HTML document than attempting to write a custom parser.

Once a webpage loads in E15, we can interact with the content by running arbitrary JavaScript code through the Python interpreter. We can interact and retrieve contents of the page by using standard JavaScript methods, or we can conveniently inject various JavaScript frameworks such as Prototype [35]. By returning values from the evaluated JavaScript back to the Python interpreter, we can use the values to augment or visually modify the 3D environment.
3. Experiments

Data Through Web Services

There are increasing number of web services that offer developers the ability to access data directly. Instead of packaging information into a designed web page, these web services provide data in a data format such as XML. Most major web services (Amazon, Google, Facebook, etc.) create web accessible APIs for various programming languages. E15 benefits from using Python; as one of the popular scripting languages used today, many APIs are available as Python libraries. With the data delivered by these services exposed to E15, we can create our own visual representations of the web content. We can also combine data from these various services and present information within the E15 environment.

By requesting data through these web services at runtime, we have access to the most recent data. This differs from other visualization tools where the data sets are static and pre-compiled; usually through scraping pages using a web crawler. The benefit of using web services rely on the fact that these services exist or is accessible to the user. As mentioned previously, many major websites include support for accessing their data through web services. However, for sites that do not offer web services, they may offer data through periodically updated RSS or Atom feeds. We can collect information provided by these feeds (or even scrape the site) periodically and store the information on our own database and create a web service to serve the data. With these capabilities, E15 is a visual web system in which current web pages can be visually augmented and API accessible web data can be graphically represented.

3.2.2 Visual Augmentation of Web Pages

One of the initial motivations for developing E15 was the desire to address some of the abilities of current web browsers. These abilities
exists primarily from the browsers' failure to evolve along the growing complexity and important nature of the Web. Current web browsers as discussed in the background section has not changed much since Berners-Lee's original web browser (§2.1).

Web browsers are designed to only deliver one page at a time. With tabbed browsing, you can load more pages without the need to open new windows; thereby keeping the screen relatively "clean." Still, you can only see one page at a time. As soon as you move from one tab to another, you will replace the view with the new page. This means that comparing information from one page with another becomes a cumbersome, and often impossible task. The user will have to open up multiple windows, and attempt to place them side-by-side or flip the page from one to the other and back again. The need for this sort of task will become even more important as more information becomes available exclusively to the Web.

Another problem that exists primarily due to web design convention, is that webpages are arbitrarily tall. With books, each page has physical limitations on how much information can be delivered – however, webpages tend to flow all information vertically. Most blogs typically exhibit this design convention, again making the task of comparing two pieces of information on the same webpage where one appears at the top and one below, awkward or impossible. It is certainly possible for an entire book to be published as a single webpage. For example, the free 348 page book; *Version Control with Subversion* comes in a single page form\textsuperscript{14}.

Keeping track of history is another difficult task. Most web browsers keep a linear track of pages that have been visited. They are usually displayed as a list, in order of most recently accessed. Because they are listed with no visual representation, we cannot visually scan for a particular page that is visually recognizable. Instead, we are forced to read and distinguish the page with just the its title. The benefits of spatial recognition

\textsuperscript{14}http://svnbook.red-bean.com/en/1.4/svn-book.html
has been previously discussed with Data Mountain (§2.3.5).

When using the E15 web browser, users can automatically render the web page as an image in the 3D environment; thus creating a visual history of all the pages that have been accessed as default behavior. Users can also manipulate the spatial position and orientation of each webpage procedurally through simple Python methods.

**Visual Filtering**

Every web page visited within E15 renders as a bitmap (OpenGL texture) image; therefore to visually modify the look of each page, we apply image filters. Since E15 is a Cocoa application developed using Apple’s Mac OS X Developer tools, it enables us to access many useful frameworks. For bitmap manipulation, we use CoreImage which is an image processing and rendering framework for Mac OS X included within the QuartzCore framework. With E15, we can use numerous image filters to achieve various effects to the web page [Fig. 3-14]; and we have the ability to change what a webpage looks like, and augment the original design.

**Levels of Detail**

Inside the 3D environment, the user has the ability to freely move around the space. Distance between the user’s viewport and a rendered page (or any other graphical element in E15) can contextualize what information from the element is relevant. Texts or images become illegible as you move farther away. Greeking is an effective visual filtering technique that is used by The Virtual Shakespeare Project, mentioned in the previous chapter (§2.3.4) and is also used in graphic design software such as Adobe Illustrator and InDesign to display the structure of the text when the text becomes too small to read. Similarly, Pad++ (§2.3.5) uses
3.2 Visual Web System

(a) nytimes.com and thenation.com displayed as crumpled newspaper.

(b) Pixellating cnn.com, bbc.co.uk and pbs.org to obscure content and bring out structure.

Fig. 3-14 Various visual filtering in E15.

...a technique in which abstract graphical representation, different from the original can be used to describe information at various scales.

The webpages rendered in E15 are rendered without any visual filters when the viewport is set at the level in which text is legible. Therefore, webpages can be read as it were designed when viewed at close distances. As the user moves away from the page, the visual effect of aging appears and can be compared with other webpages in its vicinity [Fig. 3-15]. The transition from one level of detail to another is smooth through a cross-
fading effect, to prevent any abrupt change that may disrupt the user experience and cause disorientation.

Fig. 3-15 Showing various levels of detail. Blurring decrease as user gets closer to element.

Social Highlighting

E15 also includes a drawing interface on top of the web browser. The user can use a simple highlighting tool, to highlight over information that interest them on a page. The marking will persist in their subsequent visits to the same page, thus the marking is permanent – much like marking on a page of a book. The markings from all users are merged and stored remotely through a web service over to E15:Archive (§3.2.4). The collective marking is then used to undo the effects of aging to the marked portions of the marking. The idea is to enable the collective users to highlight and identify information which they think is important on a page by page basis. We use the combined markings to create a visual repre-
3.2 Visual Web System

presentation of the web page in which areas of important information is less affected by the aging effect; and aids new visitors of the page to already seek out portions of the webpage that is more important. Result of age interruption and the drawing interface are shown in Fig. 3-16.

![Fig. 3-16 Drawing layer on top of E15's web browser enabling the user to highlight areas of interest. The aged webpage shows the highlighted areas.](image)

**JavaScript DOM Manipulation**

As mentioned previously, users can execute arbitrary JavaScript code on a webpage through the Python interpreter. This allows the user to interact with the current webpage JSON, which can be used to extract particular data that may exist on the page. This is especially powerful as the user can extract data from websites that do not offer the data through web services. In addition, access to the JSON also allow the user to modify the visual design of the webpage. This allows the user full control to experiment with augmenting the original webpage design – enabling addition, subtraction and transformation of textual and graphical elements. For instance, we can view webpages without any text, or show only the text of the page. We can also modify images or textual content. Some of
these examples are shown in Fig. 3-17.

Fig. 3-17 Manipulating images and textual content of the New York Times website. Top story reads “The Pope Challenges President; To Arm Wrestling”

3.2.3 E15:FB Visual Augmentation of Web Data

Social networks have greatly increased the number of methods we can use to interact with one another. Online interactions have become a commonplace for social interactions, and is an important everyday method for communication which rivals in importance with physical interactions. Our communications have become rich, however the way we experience these interactions is still limited – usually provided through webpages,
3.2 Visual Web System

RSS feeds and email notifications. Creating a graphical representation that visualizes the online interactions can greatly enrich the experience. E15:FB [27] is a visualization application which shows a graphical representation of social interactions with individual Facebook users. The application also provides alternative methods to navigate Facebook content beyond what is provided through the Facebook website, and creates new opportunities to interact with related content outside of Facebook.

E15:FB provides the user with visualization flexibility. The visualization is not fixed. Instead, it is a dynamic system where the user dynamically manipulate the data and how it is presented. The system can provide three types of visualization categories:

- **User query-based visualization** The user instruct what data they would like to see and how.
- **Social interaction visualization** The system can run in "watch mode," where real time events related to the user are animated.
- **Data mashup visualization** The user query additional information from services outside Facebook, to augment the existing information with additional web content such as search results, web pages, etc.

While E15:FB remains flexible enough to support any type of user-specified layout, the default layout we provide has each Facebook user represented as a cube, where the front side is occupied with the profile picture, and the left side displays the name and associated networks. The color of the name represents the cardinality of the set of networks shared with the currently logged in user [Fig. 3-18]. Videos showing the proceeding scenarios are available online\(^\text{15}\).

**User Query**

E15:FB begins by authenticating with a Facebook login, then asynchronously populating the three-dimensional environment with graphical representa-

\(^{15}\)E15 videos: http://plw.media.mit.edu/people/mud/E15FB/videos/
3. Experiments

Fig. 3-18 Showing current logged in user with friends.

...tions of the user's friends in a random position specified within a cubic region. With this state as the starting point, the user can begin executing commands on the Python console. The user can freely move each friend around in space by issuing a new coordinate, and use the two 3D mice to freely fly through the space. Additionally, a command can be executed to animate the camera to put a specific friend into view with proper orientation – this function is mapped to arrow-keys so the user can move from one friend to another through pressing the left and right arrows.

The user can also organize friends spatially. For instance, the user can execute a command that will organize the friends linearly along the z-axis – ordered alphabetically by name. Another method is to cluster friends into different regions by determining how many networks they have in common with the user [Fig. 3-19a]. This and other arrangements are simple to implement – often in less than five lines of Python code. The user can also display all photographs a specific friend appears in [Fig. 3-19b]. All photographs can be displayed in the environment at once, which greatly enhance the experience of browsing through photographs compared to viewing them individually through the Facebook site.
3.2 Visual Web System

(a) Clustering friends in groups of number of networks in common.

(b) Showing all photographs and their comments tagged with a specific friend.

Fig. 3-19 Screen shots from E15:FB.

Social Interaction

Facebook provides a notification RSS feed for every Facebook user. The notification RSS feed records all events relating to the user, providing a history of transactions within the previous week. The feed is updated as soon as an event occurs, and this provides E15:FB with real time interaction data [Fig. 3-20]. By periodically observing the RSS feed, E15:FB can visualize interactions as they occur. This creates a graphical representa-
3. Experiments

ation of the social interactions that occur within Facebook in real time, and can run on its own without user intervention.

(a) Dynamically seeing interaction through RSS notification.

(b) Inactivity with friends make them fade away.

Fig. 3-20 Facebook visualization of social interaction.

Data Mashup

As mentioned earlier, with E15's Python interpreter, we can use the vast array of Python libraries currently available. E15:FB for instance, uses the PyFacebook Python library to interface with the Facebook web API.
3.2 Visual Web System

This also means that we can use any number of other Python libraries to various other web services. For instance, we can obtain a list of a friend’s favorite bands, then make a google search request and display the results next to them [Fig. 3-21]. We can gather information from other services to make E15:FB an environment in which web data from multiple independent sources related to the user’s social network to be displayed in one environment.

![Fig. 3-21 Searching google and showing results of friend’s favorite bands.](image)

3.2.4 Implementation

E15 is a Cocoa application, created using Apple’s Mac OS X developer tools. E15 requires Mac OS X 10.5 and performance is dependent on the graphics card. We run E15 on a 8-core 3.0GHz Mac Pro running Mac OS X 10.5.2, with an ATI X1900 XT graphics card. The application performs well, even in high resolution fullscreen mode driving a 30” display at 2560x1600. In addition to keyboard and mouse inputs, E15 currently uses two 3D mice, the SpaceNavigator\(^{16}\) by 3Dconnexion to enable full 3D translation and rotation movements. The user can operate E15 without the 3D mice, instead by navigating the 3D environment with a two

\(^{16}\)http://www.3dconnexion.com/3dmouse/spacenavigator.php
dimensional scroll wheel, provided by Apple's Mighty Mouse\textsuperscript{17}. Keyboard mappings can also be overridden in the Python interpreter so that the user can move from one 3D element to another using keystrokes.

**Texture Generation and Mapping**

Currently, all graphical elements within E15 exist as a quad, with a texture map. A texture is created from a number of sources, which can be a webpage, an image file with a given URI or a bitmap created from a graphics context which can be drawn procedurally through Python. Once the texture is created, it is first resized to the closest power of two size. Then the texture is turned into a series of tiles by subdividing the texture size by 256. This is for optimization as well as to make sure the texture is smaller than the OpenGL limit; which can become an issue as webpages can turn into a very large image, from thousands of pixels to tens of thousands of pixels high. Once the textures are made, a quad is created for every texture which is then physically positioned in order. The textures are then applied, along with their mipmaps. Additional image filters may be applied (such as aging for the webpages, or blurring for higher mipmap levels) in a separate thread, and lazily loaded when the images are finally constructed.

**Web Transactions**

Much like OpenCode, E15 does not store any data on the user’s computer. We created a web application, E15:Web [Fig. 3-22] to enable saving and loading of Python scripts within E15. By saving code on the Web, the user can share and edit code written by other users of E15. The user can easily save to E15:Web by selecting *Save to E15:Web* from the File menu or equivalently pressing *command+s*. The set of Python code

\textsuperscript{17}http://www.apple.com/mightymouse/
3.2 Visual Web System

executed along with a screen capture is posted onto E15:Web, which is then stored on the site. To load a script, the user visits E15:Web with the built-in web browser, and simply clicking on the script will load the script into the Python console.

Authentication on E15:Web is done through HTTPBasic authentication. When the user first attempts to save a script, an authentication sheet drops down from the web browser, which requires the user to log into E15:Web. Once successful, the user will be able to save their script from within E15.

Similarly, the drawing data is also stored on the Web using E15:Archive. The stroke information, which exists as a Cocoa object can be serialized to a YAML file. When the user exits a page, the stroke information for that page is converted to YAML along with a bitmap image of the strokes, which are posted to E15:Archive using HTTP Basic to authenticate. When a user goes to a new page, E15 will initially load any stroke information that may exist on E15:Archive. If stroke information exists, the YAML file is parsed and converted to a Cocoa object which will then draw the strokes into the drawing layer. The stroke bitmap image will be used as a
mask to block out parts of the webpage from the aging effect.
Discussion & Analysis

In the previous chapter, I presented experiments in form of numerous web applications in addition to explorations using E15. In this chapter, I will discuss and analyze the experiments presented, by identifying the transformations that occur to these projects over time and its consequences – establishing visual and software design principles for the Web – to integrate the subtleties that exists within the physical space. I will present my discussion of the experiments in two sections. In Analysis, I define the various transformations that occur within the experiments and summarize the findings. In Evaluation, appropriate evaluation methods are discussed – especially relevant to the socially influenced Web is through participatory feedback.

4.1 Analysis

By using the experiments from the previous chapter, we begin with defining what transformations occur on the Web and where this change originates from. In Introduction, (§1.1) I mentioned the second law of thermo-
4. Discussion & Analysis

dynamics guarantees the existence of patination; however, we can relate this idea to the Web and also guarantee that patination exists within the Web as well. The second law states that for an isolated system, entropy tends to increase over time – it cannot decrease. This correlation between increase in entropy and time creates asymmetry – or preferred direction in time – which consequently explains irreversibility. The conceptual idea of the second law can be applied to the Web in order to identify the transforming time-based behavior. What is entropy on the Web? That is, what measurable quantities tend to increase?

4.1.1 Information as Entropy

The Web is an overwhelming amount of growing information. The amount of information on the Web is constantly increasing. Netcraft [1] reports that in November 2007 there were almost 150 million websites, where 7 millions sites were added within the month of November alone. Storage capacity of hard disks follow Kryder’s Law, which states that disk capacity doubles annually. Additionally, internet service providers consistently increase limits on user storage and bandwidth. It becomes more convenient to keep data online rather than having to routinely delete information, and as a result information on the Web are created much faster than they are destroyed. In essence, it takes more work to go through data and make decisions on what to delete and what not to delete.

Accumulation of information occurs every time a user requests a page from the web server – a message is appended to the web server’s log file. This log file can become arbitrarily large. For highly active websites like RunLogger, the log file over a course of a few months can grow to be hundreds of megabytes in size. During the earlier part of the Web – along with animated “under construction” graphics, web counters [Fig. 4.1.1] appeared in a vast number of webpages. Web counters informed visitors of the site on how popular the page is, given how many other people ac-
4.1 Analysis

cessed the page. Therefore, one measure for entropy is the accumulation of web information\(^1\).

Fig. 4-1 Web counters show how many people accessed the current page.

4.1.2 Age of Webpages

There is no absolute method for determining the age of a webpage. Majority of websites are now non-static pages; they are pages that are dynamically constructed based on user request. Determination of the age of a webpage becomes ambiguous. The standard HTTP method for determining the age of a webpage is to determine the modification date for the HTML file representing the page. According to the HTTP/1.1 specifications [14], HTTP response header includes the \textit{Last-Modified} field which \textit{indicates the date and time at which the origin server believes the variant was last modified}. For static files, this is typically the file system last-modified time. For pages served dynamically, this may be the most recent time within a set of included files. In the specification, it is stated that this value should be sent \textit{whenever feasible}. Therefore, some servers will not include this value within the HTTP response.

Even if the HTTP response contains the \textit{Last-Modified} field, it is impossible to know its authenticity. File modification times can easily be changed using the command line program, \texttt{touch} to any arbitrary point in time. This makes the concept of age on the Web completely arbitrary.

\(^1\)Here we are using web information as a conceptual measure of entropy, not to be confused with \textit{information entropy} in the context of information theory.
4. Discussion & Analysis

4.1.3  Time-based Behavior

We will analyze each experiment from the previous chapter and determine their social and time behaviors. Social and time factors are the main influences for transformation within the experiments. The accumulation of information, labeled as entropy, has a causal relationship with the social nature of each experiment which tends to increase. Apart from GPC, all experiments are dynamic systems – each experiment is affected by an external factor which transforms itself, and these changes create some significance.

With ever-accumulating information, there is a risk for the accumulation to become a runaway process where the amount of information can overwhelm us. We need the ability to control the information somehow – in addition to adding information – we need to be able to search, identify, edit and destroy information. By enabling various control over these features, we create an environment that can influence user habits greatly.

Another important influence in social interaction within each system is anonymity. User habits and participation are greatly influenced by whether the user's action is one that is anonymous or authored. Is there a desirability for one or the other? What are the reasons for which the user wants anonymity over authorship and vice versa.

PLWire

In PLWire, new elements are added to the site periodically. Whenever a member of the PLW decides to publicize their new project or just wants to display a photograph, video or a link – a new element is added to the site. All members of the PLW have access to add, edit and destroy contents on the site, anonymously. No one can find out who made specific changes, as change is only discovered by visiting the site. Notifications of
4.1 Analysis

change are not sent to anyone. Changes are made instantaneously, with no moderation or approval process. PLWire was designed this way from the start, as a way to create motivation for consistently updating the site with new content. This created some tension, as one member can delete another member's work without any consultation or justification.

This collaborative system can also be considered a self-centered system. Members can construct their own role. Some become adders, where their main role is to only add new information. Others become a subtractor; their primary focus is to delete information and act as the curator.

OpenCode

Like most web applications, OpenCode allows the user to edit and destroy the applets they create. For applets not created by the user, they may still tag other applets with specific keywords. Every piece of code is owned by a specific user, therefore, it is an authored system. There is a search field in which keyword searches can filter the list of graphics applets.

Filtering in OpenCode is a necessity. As mentioned in the previous chapter, there are 362 applets available. Running each of 362+ applets will be time consuming and inefficient. Currently, the applet is sorted by descending chronological order (i.e. new to old). Keyword filtering – which filters the name of the applets and associated tags – reduces the number of applets further, much like the implementations on many websites. However, the effectiveness of this filtering mechanism for users to find what they are looking for is dependent on the fact that each work is properly tagged with appropriate words.

The term authorship comes to question as OpenCode allows full disclosure of the source code for all publicly viewable applets. The user can
load the source of another user’s applet, and verbosely copy the entire code and make it theirs. The intent is to promote the idea of learning by example, but stealing of code is something that can and has been done on OpenCode. This creates another paradox in the sorting method. Since the applets are sorted from new to old, the original applet will tend to appear at the end thereby making the work that deserve credit less accessible.

We incorrectly assumed that OpenCode will be used as a convenient experimental tool, specifically for the curious, novice graphics programmer. Instead, through observation we see that most users do not create many files or make derivative works. Instead, it seems that OpenCode is used more as a showcase site where the user can present their work made with their desktop version of Processing. We came to this conclusion after a few months have past since the initial launch of the site. We saw that majority of the publicly accessible graphics programs were high quality – it was written well and in many cases, commented throughout. As each work is authored, it seems that users want to maintain a level of pride and respect for their works.

**MudSketch**

Anonymity in MudSketch allows the user to draw whatever they like without any consequences. At first glance, MudSketch is reminiscent of street walls for graffiti writers. However, I argue that this is not the case. Graffiti writers are not as anonymous as they seem. The writers take on aliases to hide their actual identity, but they do in fact use an alias identity to author all of their pieces. Graffiti writers use tags, or their personal signature to mark their pieces. They also cover public spaces with their tags, which is usually a calligraphic signature that distinguishes themselves from each other.
4.1 Analysis

Anonymity in MudSketch creates a culture more in style of a bathroom wall. The drawings are made very fast with not much thought and authorship is completely discarded – they are basically random sketches [Fig. 4-2a]. Anonymity also leads to malicious activity, beyond obscene drawings.

![Fig. 4-2 MudSketch sketch loading window.](image)

(a) Various "bathroom wall" drawings. (b) Multiple instances of the same image by one malicious user.

In MudSketch, the user saves an image by clicking the save button. There is no restriction on how many drawings you can create and save. Soon after MudSketch was launched – a malicious user took advantage of the system by clicking on the save button a few hundred times – creating hundreds of copies of one portrait drawing [Fig. 4-2b]. Behavior like this creates problems without any interesting outcome. Although MudSketch was never meant to be a curated or moderated system, there needs to be some restrictions set such that the system can still stay anonymous.

**OpenLocker and FakeID**

OpenLocker and FakeID only subtly transform over time. Both systems are OpenID services, which do not offer increase in new information. OpenLocker will add a new lock to the MIT map when a new locker is created, while a new ID will appear whenever a new FakeID account is created. These are very subtle visual changes, and are not necessarily
interesting. However, these systems have opportunities to explore other novel visual interactions.

With OpenLocker, the inside of the locker is only exposed to the user who has access to the locker. Within each locker, the user has the opportunity to create RSS feed modules and move them around the locker. The quantity of modules are limitless, however screen real estate is limited to the size of the browser window. This creates an opportunity for users to have clean [Fig. 4-3a] or messy [Fig. 4-3b] lockers. Enabling the user a choice in cleaning or keeping the locker messy is similar to Data Mountain’s use of spatial cognition (§2.3.5). When the user is freely available to move elements around, the user begins to create and impose their own set of rules on means of where certain elements are placed and why. Users are familiar with this interface within their desktop already – however, this design is not very common on the Web.

FakeID does not have much visual feedback that changes over time. As a service oriented application which serves mainly as an anonymous proxy for OpenID, the user does not see many opportunities to directly interact with the system. Currently, when a new FakeID is created, that FakeID is displayed to the top of the list of viewable FakeIDs. Any visitor to the site may view the FakeIDs, but they cannot do anything else. They only
view the FakeIDs, possibly for humor or out of curiosity. FakeID could be improved tremendously, by introducing some of the social and time behaviors that have been used in other projects.

The visual design of FakeID is unique in that to create the mood of using a FakeID, it looks rough and dirty. It is rare for a website (like most GUI systems) to be intentionally designed to look dirty. Usually UI components are clean and clear, while they try to maximize usability. The roughness only creates mood – they do not provide the user with any additional information. Physical cards, tend to degrade through usage – I look at my ATM, credit cards and driver’s license in my wallet right now, I see that the cards I use the most often are physically degraded. If IDs degrade each time it is used, the visual feedback will be meaningful by showing that degraded cards are not necessarily older, but have been used more – possibly leading to breaking the card.

Each FakeID is owned by the user who created it. Although anyone can see the FakeID, only the owner may use it. Reflecting back on this restriction, I feel that this restriction is unnecessary, and takes away the spirit of having a FakeID from the first place. If anyone can see the FakeID, they should be able to use it. With this in place, popular FakeIDs will be used more often and therefore degrade visually and provide more contrast between each FakeID.

**RunLog and RunLogger**

Although RunLog and RunLogger are driven by the same web application, they are very much two separate systems that basically do the same thing. First, the difference in success between RunLog and RunLogger can be attributed to the social network in which each belongs to. RunLog relies on itself to create its own social network. Although RunLog is an OpenID enabled system which simplifies the signup process greatly, it is
burdened by the fact that it needs to establish its own social network. A site with a simple function like RunLog should not need to establish its own social network. It is a tall order for such a small, targeted application to accomplish. However, RunLog still managed to create a network of 615 runners.

RunLogger on the other hand is wholly interacted through Facebook, as a Facebook application. There was no need to establish an independent social network, as RunLogger has accessed to the Facebook network. While being a Facebook application limits RunLogger to operate only within Facebook, this environment offers rich opportunities created by the Facebook development platform. RunLogger experiences much more daily activity than RunLog as it has a network of over 7,000 runners – the changes on RunLogger is much more frequent and dramatic than RunLog.

The RunLogger front page is the most actively transformed page. The page changes constantly, updated whenever a runner submits data. As mentioned previously – when a runner submits a new run – a news feed is created as well as the RunLogger information panel in the runner’s Facebook profile page is also updated with new statistics. The update however is represented through numbers and does not create visual contrast between individual runners. A user who has accumulated 2000 miles visually looks similar to someone who has only run 20 miles. There are important differences between the two runners when it comes to their running statistics, and there should be mechanisms in place that distinguishes them.

4.1.4 Summary: A Transformation Framework

From the analysis of previous experiments, we attempt to construct a framework in which to begin to understand how transformations occur
4.1 Analysis

on the Web. All of the experiments show that transformations occur, due to user input. Each system defined are somehow modified through user action. Given that the user is the source of transformation, we have also seen how the user is in turn affected by the individual system and its policies. Within each system, the user action fall into two extremes: Authored and Anonymous.

<table>
<thead>
<tr>
<th>User Types</th>
<th>Authored</th>
<th>Anonymous</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLWire</td>
<td>PLWire</td>
<td>PLWire</td>
</tr>
<tr>
<td>OpenCode</td>
<td>MudSketch</td>
<td>OpenLocker</td>
</tr>
<tr>
<td>RunLog</td>
<td>OpenLocker</td>
<td>FakeID</td>
</tr>
</tbody>
</table>

PLWire is unique in that it is both authored and anonymous. As previously stated, contents of PLWire is authored by members of PLW as a group. However, each member within the group has full anonymous control over each content. As a hybrid system, user activity (or more accurately, content change frequency) increases as a response to an initial change in content. For authored systems like OpenCode and RunLog/RunLogger, there does not seem to be any influence from other users' actions. This seems to suggest that within the experiments presented, systems that offer authorship do not encourage reactionary responses to other users.

MudSketch on the other hand seems to encourage anonymous users to create new drawings based on previous drawings that have been created. OpenCode, which is similar to MudSketch does not seem to promote the user to construct work based on another user's code. This could be a sign that authored environments do not necessarily provide enhancing work of another user's code due to issues surrounding originality; especially within the creative domain.
RunLog and RunLogger are both authored. However, RunLogger runners are more active than RunLog users. This is most likely due to the nature of their respective social networks. Runners on RunLog joins a brand new social network where runners may not know one another. RunLogger on the other hand exists in a pre-existing social network where runners may already have other friends using RunLogger or they end up recommending their friends to add the application. RunLogger runners seems to be more reminded to submit runs as many are active daily within Facebook.

4.1.5 The Web Through E15

Much like Pad++ (§2.3.5), E15 has been introduced as a general-purpose substrate and not an application on its own. Unlike the previous browser-based experiments, E15 enables new opportunities for experimentation with the Web that is not possible through the browser. As a development tool for visualizing web-accessible content, E15 continues to evolve with enabling new ways to visualize web data. In the previous chapter, we have seen various technological capabilities and visual explorations that have been conducted with E15. In this section, we will outline some of the consequences of using E15 and its impact on the user web experience.

Visual Representation of Age

We have seen that the Web does in fact transform over time. However, the exposure for these transformations to the user are dependent on the web designer to visually communicate these changes. E15, as we have seen can facilitate some of these transformations throughout the whole Web. E15 can visually degrade webpages to reflect the age of a webpage the user visits. By visiting a series of webpages – the user can see in the E15 environment – the difference in age of the webpages. The visual representation of aging has been experimented through image filters and ex-
4.1 Analysis

I tried to create a natural looking pattern that will give the essence of age. In the end, I came up with creating an external texture bitmap that intensified over time. The texture is multiplied onto the webpage, so as the webpage gets older, the texture becomes more dominant. This effect worked pretty well, but it was still too crisp looking. Through more experimentation, it was seen that applying varying degrees of yellowing on top of the texture creates a more intensified aged look. A sequence of a page over time is shown in Fig. 4-4.

Fig. 4-4 Aging effect over time, from left to right. Top row: 1 day, 7 days, 14 days. Bottom row: 30 days, 180 days, 365 days.

User Input

Using E15's drawing interface, the user has the ability to create markings on any webpage. The user markings made are permanent, and will affect all subsequent visits to the webpage – including other users. The drawing system is anonymous and there are no moderations in place. Once a mark has been made, the mark will be made permanent; affecting the whole E15 user community.

Anonymity is important in that the user should not be using the system
4. Discussion & Analysis

as a communication tool or a method for vandalism, though it is possible and leaving this option is an important design decision. The accumulation of markings is not a communication from one user to another, but to E15’s user community as a whole. Specific portions of the webpage which contain valuable information will be highlighted while the other portions of the webpage are obscured over time.

Limitations

With any system in development, E15 in its current state has its limitations. As mentioned previously, determination of age for each webpage (§4.1.2) is hard to determine. Currently, E15 relies on reading the Last-Modified field of the HTTP header for each page. However, many dynamically created pages do not provide this field. For instance, even if PLWre does not change its contents for a month – as the Ruby on Rails server does not report the Last-Modified field – E15 will treat the page as if the webpage is brand new. By using the command line tool, curl\(^2\), I observed that most webpages that are static html files (i.e. URL ending in html) includes the field. I have conceived an alternative solution to determining the age for webpages without the age information, by applying an image difference between two points in time. However, this is not an ideal solution as it is computationally intensive, as many webpages render as very large images.

Another limitation comes from user marking. The drawing interface was created to give the user abilities to highlight parts of the webpage – parts the user sees as useful information. This however leads to many issues. It is naive to think that all users will cooperate and use the highlighting tool sparingly. The metaphor used for the site was to collectively and anonymously mark pages, such as marking books from a library or a used textbook that exchanges from one student to another. But this literal

\(^2\)The command curl -I [URL...] will return only the HTTP header of the response.
4.1 Analysis

mapping will not work, as potentially massive amount of users can mark a single page and overfill the whole page – rendering the marking useless.

There are ways in which we can work around this issue. One method is to make the markings only effect the users that originally created the marking. However, this will destroy all notion of a socially influenced system. Another method would be to select a group of people who you want to share markings with. This system will then take away anonymity of the system, which will affect the community, much like OpenCode (§4.1.3). The system could also take a weighted average of all the markings. This will filter out any possible noise and only leave portions that are collectively agreed as important. The final solution can be a system where markings disappear over time. The markings can either gradually fade or disappear completely after a certain time period. Even with these potential solutions, it is hard to determine the outcome until we have a wide-scale user community. For now I feel there should be flexibility in the system such that any user can decide to mark the entire site.

E15 offers various other methods to modify the visual outcome of webpages. This can be done independently by the user. As previously detailed, E15 can modify the visual elements of a webpage by interacting directly with the DOM using JavaScript passed through the Python interpreter. This enables the user to explore ways to augment the original webpage, and using E15:Web (§3.2.4), share the code with others. Of course, the limitation in this method is that it assumes prior knowledge of JavaScript and DOM manipulation. In general E15 is a development platform created for developers and not for end users.
4.2 Evaluation

Effective evaluation methods are required to assess the successes and failures of the work in this thesis. However, determining the effective means of evaluation is a hard challenge on its own, given the multidisciplinary nature of this thesis. Like many theses from the PLW, the work presented in the thesis cross multiple disciplines, spanning from design to software engineering of utilities to creative tools; resulting in its own artistic endeavors. It is not necessarily the usability issues – the success of the work in this thesis is ultimately defined by how it may influence future work on the Web.

4.2.1 Participatory Design

Participatory design strategies suggest that more user involvement helps determine information about user tasks, in addition to creating opportunities for users to influence design decisions. Users become part of the recursive design cycle. This kind of design strategy points to the importance of users – not only as the user of the software but also as an influence for future development. Many web applications – with their developers sharing development plans with their users – employ this strategy which reiterates the importance of the user.

This is much contrary to traditional software companies that keep development ideas confidential rather than a continuous dialog with their user community. Of course there are criticism to the participatory design approach, as extensive user involvement may be costly and complicate implementation. It could also create some antagonism from uninvolved users or users that feel ignored by developers for not implementing their ideas.

RunLogger has been heavily participatory in its development, and this
4.2 Evaluation

participation from the users motivates its continued development. Through comments published on a public page, users provide feedback on how to improve current functionality or they offer praise or criticism [Fig. 4.2.1]. I use continued feedback from users to inspire what new feature to implement. These users will also send bugs and report server errors. For example, I was sent a Facebook message regarding an error received by a RunLogger user alerting me of not being able to load the application. It turned out that our production server suffered a hardware failure. Without this user alert, it may have been hours or days before any one of us in the group realized that there were anything wrong with our production server.

Fig. 4-5 RunLogger users post comments and suggestions in a public discussion board.
Conclusion

The Web is a dynamic, social, enigmatic world much like the physical world. The thesis is an initial investigation into understanding the intricacies of the Web which has its own analogous similarities to the physical world, yet enabling us to create new opportunities that do not currently exist. As we spend more time replacing interactions from the physical to the Web, we begin to lose some of the enjoyable qualities of physical things. This thesis attempts to show that the Web itself has opportunities to create a rich environment with their socially influenced, time-based behavior.

5.1 Conclusions

As we have seen from the experiments conducted, it is the user which acts as the influence that can transform the Web. The everyday transformations that occur on the Web is caused by user interaction. From the thesis we have seen that the users themselves are influenced by how each web systems are implemented – what each user is allowed to do, and
whether they are responsible for their behavior. Through the experiments presented, we see that the Web has its own set of subtleties that cannot be experienced in the physical world.

To enable new explorations, we began development of E15 as a way to interact with the Web outside of the web browser. As a result, we created a general graphics framework for web-accessible content. With the ability to interact with multiple webpages and variety of web services, we can create new experiences that enables the user full control of all web-accessible information.

As a web designer, the possibility of designing for the Web using a tool like E15 is exciting. You no longer publish a "page," instead creating and publishing a visual environment constructed using resources available on the Web, while also enabling various interaction methods. This is much more powerful than many of the Web mash-up ideas that are currently available. However, the principle of this system is very similar to the Berners-Lee Web. The source code for each of the visualization exist as human-readable Python code – just like webpages are constructed using a collection of human-readable HTML, JavaScript and CSS files. This allows new developers the opportunity to see the work and understand how the code translate to what they see on screen. I believe this feature is of utmost importance and should be preserved as the standard of the Web.

5.2 Future Work

It is impossible to predict how the Web will evolve in the future, as the Web does not have a defined goal. It is open to motivated individuals and collaborative groups to define future directions. I see the work by PLW through E15 in the past year as an exciting initiative – as an initial
5.2 Future Work

exploration of ways to interact with the Web in which users begin to collectively and collaboratively augment information from all parts of the Web. The Web itself becomes a dynamic environment where information – through social influence – transform over time. In essence, this destroys the notion of a web designer. However, I do not see it as destruction – instead, a redefinition of what we currently acknowledge as the standard of web design.
Bibliography


