On Mobile Accessibility: Learning from our Desktop Ancestors

by

Darren Minifie
B.Sc., University of Victoria, 2008

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

On Mobile Accessibility: Learning from our Desktop Ancestors

by

Darren Minifie
B.Sc., University of Victoria, 2008

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. George Tzanetakis, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. George Tzanetakis, Departmental Member
(Department of Computer Science)

ABSTRACT

The art and science of creating effective mobile software has become significantly more important since on-the-go computing has taken centre stage in users' daily lives. This area of application development introduces many new constraints not commonly found in mainstream desktop computing. Misunderstood user requirements, limited hardware resources and reduced software services all contribute to an environment foreign to many developers. In its infancy, accessible desktop computing faced many of these same constraints. Individuals with disabilities generated unconventional user requirements, hardware specific to accommodating the disabled had yet to be developed, and third-party assistive services were not implemented. As such, developers were forced to create unique programs and user interfaces to deliver effective solutions. In this thesis I propose that the creation of mobile software should draw from that which was learned from the creation of its accessible desktop predecessors.

To support this claim, I present four case studies. These examples include a system to automate the lookup of transit scheduling information, a music collection browser, a framework for presenting flowchart-like data structures, and a language learning tool. In each case study, an accessibility-related problem is solved on the desktop without the reliance of third-party assistive support. Therefore, it follows that these applications translate to mobile platforms much more cleanly than applications dependant upon adaptive services. This claim is validated by evaluating a prototypical mobile implementation of each example. This evaluation implied a positive correlation between accessibility-minded desktop applications and their general-purpose mobile counterparts. Based on these results, I believe it is in developers' best interest to consider accessible desktop applications as reference implementations when designing effective mobile software.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank the following people and organizations for their contributions and support:

- Celina Gibbs for unpublished work related to computer science education and language learning techniques.
- Chris Pearson for unpublished work related to platform agnostics, portable software, the Symbian operating system and Flash Light.

Chapter 1

Introduction

A user interface (UI) is the gateway between the user and a computer. Through this gateway the user provides her intent via instructions, and receives feedback from the computer as a result of executing that intent. For this reason, two-way communication between the computer and the user is the cornerstone of a good UI. Specifically, desirable traits of any UI include: clarity, consistency, simplicity, controllability, directivity, forgiveness, feedback-awareness, efficiency and aesthetics. [1]. Although the desirable traits of a UI are fairly well agreed upon, design models and concrete implementations vary widely. The most traditional graphical user interfaces (GUI) have many widely accepted features including such things as keyboard input and shortcuts, mouse point-and-click, drag-and-drop, mouse-manipulated control widgets, tabular presentation of information, and menu-driven functionality to name a few. An implementation consisting of these established components is common, but does not fit all use cases.

Mobile computing is fast becoming a primary means for information consumption and productivity. As a reaction to users' demands for quick and brief interactions with their smartphones, mobile applications are more targeted and specific to a single task in comparison to their desktop counterparts. Further, the ways in which users interact with mobile applications differ vastly from desktop computing. For example, mobile hardware often employs multitouch as its primary source of input, display sizes are significantly reduced and processing power is limited. For these reasons, application developers must find alternative means to creating software with these new constraints and usage trends. For many developers, this shift in UI design has been a struggle; many people have insisted on porting desktop UIs - an approach that is ultimately not viable.

In this thesis, I propose that application and platform developers need not start from scratch when creating end-user applications and UI frameworks, respectively. Coping with

hardware limitations and unconventional user requirements has long been the practice of accessible software development. Regarding desktop computing, in many cases, users with disabilities rely on third party services to augment mainstream applications to improve accessibility. For a variety of reasons, this approach of usability mitigation is not always appropriate. For instance, third-party services simply may not exist (as is the case with less popular operating systems), suitable hardware and drivers may not be present to support alternate methods of user input and computer output, or the necessarily generic approach to third-party assistive support is insufficient to satisfy niche user requirements and domain applications. Accordingly, desktop application developers have taken it upon themselves to create highly specific programs that operate and interface with users in some unique fashion.

I have observed that mobile technology in its current state is subject to many of the same constraints as accessible desktop technology. It should, therefore be possible for developers to employ similar techniques when creating mobile applications as were used to create their accessible desktop counterparts. To support this claim I present four case studies. In each case, a problem related to accessibility was solved on the desktop without relying on third-party assistive support. Table 1.1 outlines the domain-specific characteristics of each case study. Table 1.2 outlines the accessibility challenges associated with each. These applications, therefore translate to mobile platforms much more cleanly than applications dependant upon adaptive services.

	Transit Information Viewing	**Music Collection Browsing**	**Flowchart-Like Data Structures**	**Language Learning**
Example implementations	static HTML tables, rich clients (e.g Google Transit)	media players (e.g. Apple iTunes)	productivity suites (e.g. Microsoft Office), creation tools (e.g. OmniGraffle)	thick clients (e.g. Rosetta Stone), Adobe Flash content
Applicable assistive services (software)	screen magnifier, screen reader, onscreen keyboard			*vendor implemented*
Other software adjustments	custom user CSS, high-contrast default colours, large default fonts	high contrast system colour theme, large default fonts, low resolution	high contrast user-selected colours, large fonts and figures, low resolution	*vendor implemented*
User input methods	mouse, keyboard, voice, eye-tracker, automated scripting	mouse, keyboard, voice, eye-tracker, automated scripting	mouse, keyboard, automated scripting	*vendor specific*
Output methods	monitor, braille, printer	monitor, braille	monitor	monitor

Table 1.1: Characteristics of accessible desktop software

	Transit Information Viewing	**Music Collection Browsing**	**Flowchart-Like Data Structures**	**Language Learning**
Conflicts with screen magnifiers	lose context without table headings, Javascript fly-out menus not viewable	lose context without table headings, small control widgets difficult to find	tedious panning required to see full figure, tooltips near edge aren't viewable	UI updates not visible outside of zoomed-in area, tooltips near edge aren't viewable, contention for graphics resources
Conflicts with screen readers	unvalidated HTML markup confuses reader, generic markup conveys no semantic meaning, style and content mixed confuses reader	cell-by-cell readout is fatiguing, readout audio is masked by song audio, semantics of graphics is lost (e.g. album art view)	graphical figures difficult to represent with audio, widget readout depends on hooks for reader, information based on colour is not conveyed	all / any readout depends on hooks for reader, graphical information difficult to represent with audio, foreign language may be unsupported by reader
Conflicts with user input	keyboard mappings altered by browser, user must navigate through irrelevant content	keyboard-based navigation is tedious	heavily reliant on mouse point and click, drag and drop, extensive switching between tool palettes and canvas, widget selection is fine-grained	no standards exist for input methods, keyboard shortcuts are non-standard, keyboard shortcuts may map over standard keys

Table 1.2: Challenges associated with accessible desktop software

1.1 Thesis Overview

This thesis begins with Chapter 2 which provides context for the current state of accessible and mobile computing. Here I provide general information on usability and accessibility, a survey of mobile hardware and software, and domain-specific material whose potential impact on accessible computing is significant.

Chapters 3, 4, 5 and 6 detail four projects that provide support for my thesis argument. Chapter 3 (BUSSPASS) describes a Web / mobile based application and framework whose purpose is to facilitate the retrieval of specific public transportation information from unmanageably large data sources. MarGrid Mobile, an assistive music browser, is discussed in Chapter 4. While many multimedia applications rely on textual interfaces to convey media information to the user, MarGrid Mobile employs Music Information Retrieval (MIR) algorithms and a tactile interface that allows the user to browse his music collection by touch. Chapter 5 describes an alternative approach to the graphical creation of information networks (for example, flowcharts) using an implicit patching model. Finally, Chapter 6 describes WIKSSI: a multimedia, community-driven tool that supports a dynamic alternative to more static learning methods.

Based on these four case studies, Chapter 7 organizes common and significant features into an abstract framework that supports the design and development of accessible applications. Chapter 8 provides a discussion of these use cases, and their varying conformance to the aforementioned framework. Finally, Chapter 9 summarizes the analysis and evaluation of this work, and proposes an avenue of future study.

Chapter 2

Background and Related Work

2.1 The User Interface

The user interface, also known as the human-computer interface, is the threshold of interaction between a computer and a user. This interaction is bidirectional, consisting of user input and computer output. The user expresses her intent to the system by providing input which controls and manipulates a running program. The user receives feedback to her actions from the system in the form of computer output. Input and output vary widely in their form, including keyboard, mouse and touch for the former, and display, printer or audio for the latter.

Many types of user interfaces exist including text-only *commandline* interfaces, mouse-based point-and-click *graphical* user interfaces, gesture interfaces and voice interfaces.

As the number of computer users increases, so too does their demand for simple and efficient technology solutions. Businesses rely on modern computer systems to increase employee productivity, while at the same time expecting to pay little for training. The problem from a developmental perspective is that although user interfaces have become simpler for the user, they have also become more complicated to create [2].

2.1.1 Human Computer Interaction

Human Computer Interaction (HCI) is defined as "a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them " [3]. It is an interdisciplinary field of study involving both sides of the user interface, namely the machine side and the human side. Machine-side factors include programming languages, computer graphics and oper-

ation systems, while human-side factors include cognitive psychology, social sciences and human factors.

Closely related to HCI are the fields of ergonomics and human factors. While somewhat overlapping, ergonomics focuses on the relationship between the user and his surrounding system with the goal of minimizing physical injury. "Human factors" is a more general area of study, and HCI can be thought of as human factors specifically related to the interaction between computers and users.

One major outcome of HCI is to map, as closely as possible, the human cognitive model to the computer's understanding about what is to be accomplished. The means to this end include user interface comparison and evaluation techniques, development of new interaction paradigms, refining user interface design methodologies, and building upon predictive user interaction models.

Among the most highly regarded HCI design principles and methodologies include empirical measurement and iterative design.

2.1.2 Usability

Usability refers to the ease at which a user can accomplish some task using a particular tool. In the computer science (CS) and HCI domains, this usually translates to the clarity and efficiency of the user interface. Measuring usability is difficult because it is highly qualitative. Attributes such as ease of learning or level of user frustration must be used judiciously when drawing conclusions due to their subjective nature.

Neilson and Shniederman identify five factors of usability. First is learnability, which relates to how easy a user interface is to use when accomplishing some task upon first encounter. Closely related to this is memorability, or the degree to which users remember how to use a system on a repeating basis. Next are efficiency and errors. Efficiency refers to how quickly users can accomplish some task while errors represent the mistakes made when achieving those tasks. Finally, satisfaction refers to how enjoyable the system is to use [4, 5].

Associated with usability are usability testing and usability engineering. The former is a measure of the ease of use of a UI while the latter is the design and development process used to ensure a highly usable UI.

2.2 Accessibility

Closely related to the field of usability (see Section 2.1.2) is that of accessibility. While usability refers to the degree to which a system is usable to a specific target user base, accessibility refers to the overall access to a system by as large an audience as possible. Accessibility is commonly associated with the degree to which people with disabilities can access a system.

Directly addressing accessibility issues involves the development of user interfaces that accommodate specific user groups. Indirect accessibility support is achieved by delegating responsibility to third-party adaptive software through hooks within the system (for example, scripting support).

The disability rights movement [6] has produced important guidelines and pieces of legislation relating to accessibility and technology. At the forefront is the World Wide Web Consortium (W3C) [7] with the introduction of the Web Accessibility Initiative (WAI) [8]. The WIA is an initiative whose mandate is to improve Web accessibility for people with a disability. It is comprised of working groups, interest groups and task forces who produce guidelines and technical reports on a spectrum of Web technologies, from development and content creation tools to user agents and rich internet applications (RIA). These various guidelines are aggregated into five formal recommendations: the Web Content Accessibility Guidelines (WCAG) [9], the Authoring Tools Accessibility Guidelines (ATAG) [10], User Agent Accessibility Guidelines (UAAG) [11], the XML Accessibility Guidelines (XAG) [12], and the Accessible Rich Internet Applications Guidelines (WIA-ARIA) [13]. Together, these documents serve to both improve the Web experience for people with disabilities, as well for users with non-standard devices such as mobile phones.

Voice Extensible Markup Language (VoiceXML) is a standard designed to model audio response conversations involving both synthesized audio and actual audio samples [14].

2.3 Mobile Computing

Mobile computing, which may be thought of as the consumption of technology ‘on-the-go’, is an increasingly popular trend in technology in recent times. Mobile computing goes beyond conventional portable computing, a paradigm that forced users to temporarily remain stationary while using their technology (laptops for example).

Mobile devices have empowered users in many ways, providing quick access to content, communication and tools. The range of services range from the viewing of simple

static documents, to accessing the Web, to Augmented Reality. Augmented Reality (AR) is the combination of real world views with computer generated views [15]. A popular implementation sees a mobile device screen showing a real view of the surrounding environment (via its camera) overlaid with supplemental information about those surroundings retrieved from the internet.

Although mobile devices expose many new avenues for computing, they are not without their limitations, some of which include weak security protocols, limited power reserves, insufficient access to internet bandwidth and potential health concerns. Another shortcoming related to accessibility and usability is the user interface. The necessarily reduced form factor has several ramifications for people with disabilities. Reduced display sizes limit the screen real estate available for content presentation. Small keyboards and touch displays create input problems for people with motor and vision disabilities respectively. Exacerbating these problems are underpowered processors incapable of providing sufficient resources for assistive technologies such as screen magnifiers, screen readers and voice recognition.

Because the market for mobile technology is so vast, several product tiers have developed. The most basic and inexpensive devices are capable of making calls and sending text messages (SMS). The middle tier, referred to as *feature phones* provide additional functionality primarily though embedded applications. Finally, the upper tier, or *smart phones*, are the most advanced devices. They usually contain a variety of sensors, radios and other methods of input / output (IO), and usually run sophisticated operating systems capable of supporting third-party applications and services. While expensive, smart phones represent the future of mobile technology and will be the focus of this thesis.

2.3.1 Mobile Technology Survey

Mobile computing is among the fastest changing areas of technology in terms of evolution and refinement. Hardware manufacturers are continually introducing new handsets with faster processors, larger memory, longer lasting power reserves and new sensors. Software vendors constantly push this hardware to its limit with sophisticated operating systems, aesthetic user interfaces, and highly functional applications. Table 2.1 summarizes the current state of mobile hardware by providing a comparison feature vector for a sample of the most popular mobile handsets.

	HTC HD2	**Motorola Milestone**	**Nokia N900**	**RIM Blackberry Bold 9700**	**Apple iPhone 3GS**
OS / Platform	Windows Mobile 6.5	Google Android 2.1	Maemo	Blackberry OS 5.0	iPhone OS 3.1
Processor	GHz Qualcomm Snapdragon	ARM Cortex A8 600 MHz	ARM Cortex A8 600 MHz	624 MHz	ARM Cortex A8 600 MHz
Memory	448 MB	256 MB	256 MB	256 MB	256 MB
Storage	microSD, microSDHC	16 GB on microSD, microSDHC	32 GB internal, microSD	256 MB internal, microSD	32 GB internal
Display	4.3 inch, 800 x 400 resolution	3.7 inch, 854 x 480 resolution	3.5 inch, 800 x 480 resolution	2.44 inch, 320 x 480 resolution	3.5 inch, 480 x 320 resolution
User Input	capacitive touchscreen, onscreen keyboard	capacitive touchscreen, physical keyboard	capacitive touchscreen, physical keyboard	physical keyboard	capacitive touchscreen, onscreen keyboard
General Communications	Bluetooth, Wi-Fi	Bluetooth, Wi-Fi	Bluetooth, Wi-Fi	Bluetooth, Wi-Fi	Bluetooth, Wi-Fi
Cellular Communications	GSM/EDGE, HSDPA	CDMA, EVDO	GSM/EDGE, HSDPA	GSM/EDGE, UMTS	GSM/EDGE, UMTS/HSDPA
Navigation	GPRS	GPRS	GPRS	GPRS	GPRS, digital compass
Camera	5.0 megapixel	5.0 megapixel	5.0 megapixel	3.2 megapixel	3.0 megapixel

Table 2.1: Comparison of mobile hardware

This table illustrates several important trends among the most pervasive smart phones. First, several features are common across the entire set of handsets including GPS, Wi-Fi, Bluetooth and a camera. This consistency allows developers to plan for multi-platform applications. However, portable software is hindered by the operating systems, APIs, development tools and programming languages that vary from device to device. Computing resources, specifically processing power and memory, are also limited relative to their desktop counterparts. While desktop memory commonly amounts to gigabytes and CPUs operate at over three gigahertz, mobile RAM is typically limited to 256 megabytes, and mobile processors operate at around 600 megahertz.

Directly relating to the user interface is the display and method of user input. A recurring theme of this thesis is the reduced size of the mobile display and the adjustments that must be made to produce effective UIs. At the lower end of the spectrum is the Blackberry with a screen size of 2.44 inches and a resolution of 320x480 pixels. While this display is minuscule, the trend in mobile screen sizes is moving to larger form factors. A more common size is that seen on the Motorola Milestone, which has a screen size of 3.7 inches and a resolution of 854x480 pixels. In landscape mode, the horizontal resolution is approaching that of lower end desktop displays. The Blackberry is also the only device that relies exclusively on a physical keyboard for user input. While several devices include an optional physical keyboard, the primary form of input is a capacitive touchscreen. Several devices also implement haptic technology to provide tactile feedback to the user. For example, touching a widget on a touch screen results in the device vibrating.

Ubiquitous connectivity is also common among devices. Wireless networking is available on all devices via Wi-Fi, and 3G cellular connectivity. Unfortunately the problem with the latter is the competing standards and technologies in cellular technology. High-Speed Downlink Packet Access (HSDPA) which is based on Universal Mobile Telecommunications Systems (UMTS) [16] and GSM (Global System for Mobile Communications) [17] technology is in direct competition with Code division multiple access (CDMA) [18]. While GSM is more accepted world-wide, CDMA is a significant competitor in North American markets. Therefore, handset manufacturers are faced with difficult decisions about which technology to implement, often leading to consumer confusion and higher handset prices for hybrid hardware.

2.3.2 Software

Development of analysis tools to evaluate accessibility for mobile applications is complex due to the wide range of UI architectures in existence. This section surveys the UI architectures for three widespread platforms: iPhone, Android and Symbian.

iPhone OS

The original iPhone and iPhone 3G run a reduced version of OSX called iPhone OS. Though the software development kit (SDK) for this platform is a fraction of the SDK available on the Mac, the architecture of an application is very similar. Both platforms make use of Apple's core framework: Cocoa[1]. Cocoa is an umbrella framework, providing references to all of the technologies available to Mac/iPhone developers.

iPhone UIs are implemented using the classes of Cocoa's UIKit framework. As part of the development environment, Apple provides a "what you see is what you get" (WYSIWYG) editor for the creation of UIs called Interface Builder.

Though UIs can be developed programatically, a more common approach is to use xib files. A xib file is an XML representation of several UIKit objects that have been pre-instantiated and are loaded at runtime [19].

When it comes to accessibility evaluation, this application architecture has several implications. Static analysis of source code may or may not provide useful information about the user interface, depending upon whether the developer decided to implement the UI programatically or using Interface Builder and xib files. If the UI is implemented programatically, static analysis tools can extract relevant accessibility properties from the View and View Controller classes. However, if the UI is configured with external xib files, other analysis approaches may be necessary. For example, dynamically inspecting View objects as they are instantiated at runtime would allow for an evaluation that detects many accessibility problems that would otherwise be undetected because of the use of runtime configuration files. That said, dynamic analysis is a complicated process in its own right, and would likely be challenging to implement in a platform-agnostic fashion. However, another form of static analysis may be possible; analysis of the configuration files themselves. xib files (as well as Android resource files, see Section 2.3.2) are described using XML. Tools that can parse these types of files, and analyze the various attributes and elements, would surely be effective in finding potential accessibility problems. In fact, such an approach is heavily underway in the Web accessibly domain. The W3C has provided a

[1] Legacy Mac applications use the older Carbon framework

complete listing of tools to evaluate the accessibility of Web pages [7]. Many of these tools are open source, and could be modified to check for certain accessibility problems specific to these types of resource files.

Android

Android is an open source platform from Google targeting a variety of mobile devices. Specifically, it is a software stack consisting of a Linux kernel, Dalvik VM, core libraries and an application framework [20]. Applications are written using the Java programming language; however, the program is not compiled to standard Java bytecode. Instead, the included *dx* tool compiles the program to a format suitable for executing on the custom Dalvik VM.

The Android UI architecture is quite similar to that of iPhone OS (see Section 2.3.2). UIs can be implemented programatically by subclassing View classes and controlling them with Controller classes (Android refers to these as *Activities*), or through external resource configuration files which are XML based.

Symbian

Symbian Series 60 (S60) has taken a more open stance on the programming languages used to implement native applications. C++ is the formal language; however, officially supported runtimes exist for both Java ME and Python. Python for Symbian (PyS60) is very robust and well supported - so much so that some commercial Symbian applications are written in this language.

The Symbian Foundation, as well as Nokia and other manufacturers of Symbian phones, are very active in creating developer tools. Carbide C++, the de-facto development environment for native Symbian applications, is a world-class Eclipse-based development environment that is made available at no cost.

Flash Lite

Adobe's Flash Lite is another option for developers who wish to deploy applications with significant cross-platform support. Flash currently runs on Symbian Series 40 and S60, as well as Windows Mobile devices, and is expected to be ported to Android and BlackBerry by the end of 2010 [21]. Unfortunately, Flash support for the iPhone platform is not expected.

In the desktop world, Flash is known for creating interactive web sites and applications that work in the same way on any desktop OS. Flash Lite brings a subset of these features to mobile devices, while considering critical issues such as slower processing speeds and small storage capacities [22].

Silverlight For Mobile

Microsoft's recent challenger to the desktop dominance of Adobe's Flash, Silverlight, is poised to bring its own cross-platform support to mobile devices. Silverlight For Mobile brings the promise of consistent application look and feel across platforms, along with a standardized API that should allow an application to "work across different devices and platforms" [23].

As of this writing Silverlight For Mobile has not been released to the public, but Microsoft has stated that their product will initially be available on Windows Mobile and Symbian S60 devices.

2.4 Machine Learning: Artificial Neural Networks and Self Organizing Maps

Machine learning is an indispensable tool when it comes to creating efficient systems that require less manual user intervention. By providing adequate data, machine learning algorithms allow the system to make informed decisions and accurate estimations. With the system able to automate more tasks, UIs can be simplified, leading to higher usability. Arthur Samuel defined machine learning as follows:

> A computer is said to learn from experience E with respect to some task T and some performance measure P if its performance on T improves with E [24].

There are three broad categories of machine learning: supervised learning, unsupervised learning and reinforcement learning. Supervised learning attempts to define a function that predicts an output based on some given input. It does this by learning from a training set consisting of inputs and their true outputs. When the resulting function predicts a continuous value the process is called a regression; when the function predicts some quantized class the process is called classification. Unsupervised learning differs in that predetermined outcomes within the data are not known. Unsupervised learning attempts

to find patterns and organization within unlabelled data. A popular unsupervised learning technique is clustering. Reinforcement learning attempts to map environment state to the actions an agent must make in that state. The objective is to maximize some reward based on the sequence of actions taken by agent.

An artificial neural network (ANN) is a computational model that attempts to achieve learning by simulating the behaviour of a biological neural network (NN). It is made up of a set of nodes which may have one or many connections to neighbouring nodes. Data is fed into the network via one or more input nodes, is processed by a connected graph of internal nodes, and output by a set of output nodes. ANNs are a powerful learning technique because their structure can change based on the information they consume during the learning process.

A self-organizing map (SOM) is a type of artificial neural network based upon unsupervised learning techniques. During training, the input data (which can be of high dimensionality) is mapped to an output space of a low dimensionality - usually 2 dimensions, representing a grid. Input data is quantized, called vector quantization. The mapping stage then classifies the input samples, mapping them to the lower dimension output space. This reduction in dimensionality is desirable for 2D and 3D information visualizations.

Chapter 3

Case Study I - BUSSPASS

Public transit is an essential service for many residents of urban areas. People rely on this service for business and personal use. A major factor in the efficiency and quality of its use; however, is scheduling information. The traditional approach to presenting transit scheduling information is to use a table. An example is shown Figure 3.1. Columns represent the various stops along a particular route, and rows represent the various departure times for each stop. Each table is identified by a combination of the current weekday, direction and route.

The most typical medium for such a table (aside from printed hard copy) is HTML. Transit authorities post these HTML documents on their websites for their customers to consume. The problem with this form of presentation is the sheer amount of data. The tables attempt to encapsulate many variables including the day of the week, time of day, direction of travel, and route. Such an information scheme begs to be searchable rather than browsable; however, in its table-based incarnation, the onus of information reduction is placed on the user.

Route 7 - UVIC / GONZALES / DOWNTOWN EAST

(Effective **Tue Sep 8, 2009** through **Sun Dec 27, 2009** inclusive)

View Printer-Friendly Version View Map

Choose Direction: EAST | WEST

Choose Day: Monday through Friday | Saturday | Sunday

Monday through Friday - Morning

Bus Type	Douglas at View	Fairfield at Blanshard (Blanshard Terminus)	May at Moss	Crescent at Quimper	Fairfield at Moss	Foul Bay at Fort	UVic Exchange
Access	6:42	6:45	-	-	6:50	7:00	7:12
Access	6:58	7:01	-	-	7:06	7:16	7:28
Access	7:15	7:18	-	-	7:23	7:33	7:45
Access	7:28	7:31	-	-	7:36	7:46	7:58
Access	7:40	7:43	7:50	7:58	-	8:05	8:17
Access	7:50	7:53	-	-	7:58	8:08	8:20
Access	8:02	8:05	-	-	8:10	8:20	8:32
Access	8:15	8:18	-	-	8:23	8:33	8:45

Figure 3.1: A typical public transit schedule

When it comes to a UI for such a table, screen real estate is of utmost importance. A wide display can accommodate more columns for location-based information; a tall display can accommodate more rows for temporal-based information. Table headings provide an essential context for the interior data. Without the location information provided by headings, row entries lose their meaning. This constraint is significant for both accessible and mobile computing. If the user is forced to zoom in on a particular area of the table, he loses the location context provided by the table headings, see Figure 3.2

6:42	6:59	7:07	7:11	7:20
6:52	7:09	7:17	7:21	7:30
7:02	7:19	7:27	7:31	7:40
7:12	7:29	7:37	7:41	7:50
7:22	7:39	7:47	7:51	8:00
7:32	7:49	7:57	8:01	8:10
7:42	7:59	8:07	8:11	8:20
7:52	8:09	8:17	8:21	8:30
8:02	8:19	8:27	8:31	8:40
8:12	8:29	8:37	8:41	8:50
8:22	8:39	8:47	8:51	9:00
8:32	8:49	8:57	9:01	9:10
8:42	8:59	9:07	9:11	9:20
8:52	9:09	9:17	9:21	9:30

Figure 3.2: A zoomed-in transit schedule, as it might look on a mobile device or magnified screen

In many cases, the static Web page that displays transit information is the only digital source of data provided by the transit authority. This is problematic for several reasons. First, the technology that generates the HTML often does a poor job, producing non-validating markup. Web Accessibility software often relies on valid and complete HTML to provide an accurate alternative of the content to its users. An out-of-fashion technique in Web design was to use `table` tags to lay out documents. While this produced a desirable layout, table cells used for spacing had no relation to the document contents, and had no semantic meaning. As such, assistive tools were unable to determine which document elements represented actual content, and which were for style. Second is the difficulty in extracting content to be used in other presentation schemes. Generally, when an organization wishes to share its content with developers, it exposes an application programming interface (API). The API is used by third parties to efficiently request the data they require. It also promotes good communication between parties because of the formality of the API. Without an official API, developers are forced to use screen scraping techniques. Screen scraping is the process of acquiring data by accessing Web resources in an end-user fashion, then extracting and transforming that data with customized processing software. It is

viewed as unethical by some because it attempts to forcibly steal content without permission. However, it is more acceptable in scenarios where data is inherently public. Screen scraping is less desirable for all parties involved. From the perspective of a third party developer, screen scraping requires significantly more effort to implement and resources to execute. Further, if the structure of a provider's content changes, screen scraping algorithms must be adjusted to avoid breaking compatibility. From the provider's point of view, resources are wasted by third parties uncontrollably connecting to their servers to scrape data.

Services such as Google Transit [25] attempt to alleviate this problem by aggregating transit scheduling information from many providers and presenting it in a unified fashion. Google has defined a data schema that providers must adhere to. Once the schema is implemented, an RSS feed is created which continually feeds Google's service with the most recent data. Unfortunately Google has not yet made this unified data available to third party developers.

Transit organizations are beginning to provide scheduling information using other mediums. For example Translink (Vancouver's transit provider) provides an SMS service to its users. The user begins by texting a short message to the number 3333. The short message contains a five-digit code that uniquely identifies the stop at their current location. An SMS message is returned displaying the next five departure times for that stop [26]. Accessibility problems exist with this solution as well. For example to be able to send the unique five digit code, the user must be able to read it from a bus stop. This would not be possible for people with vision disabilities.

A significant challenge to using public transit is managing the vast amount of scheduling data in an efficient manner. This information bloat is attributed to several dimensions including day of week, time of day, city, travel direction, transit route and departure time. As depicted in Figure 3.3, a typical schedule attempts to include and present all of this information in one or more large documents. Increasingly large documents become unusable by users who rely on assistive technology or mobile devices.

A better approach is to use readily available technologies to automatically eliminate as many dimensions as possible, thereby greatly reducing the amount of information that must be manually inspected by the user. The application and integration of these services, as well as a means of deployment, together constitute a system we call BUSSPASS [27, 28].

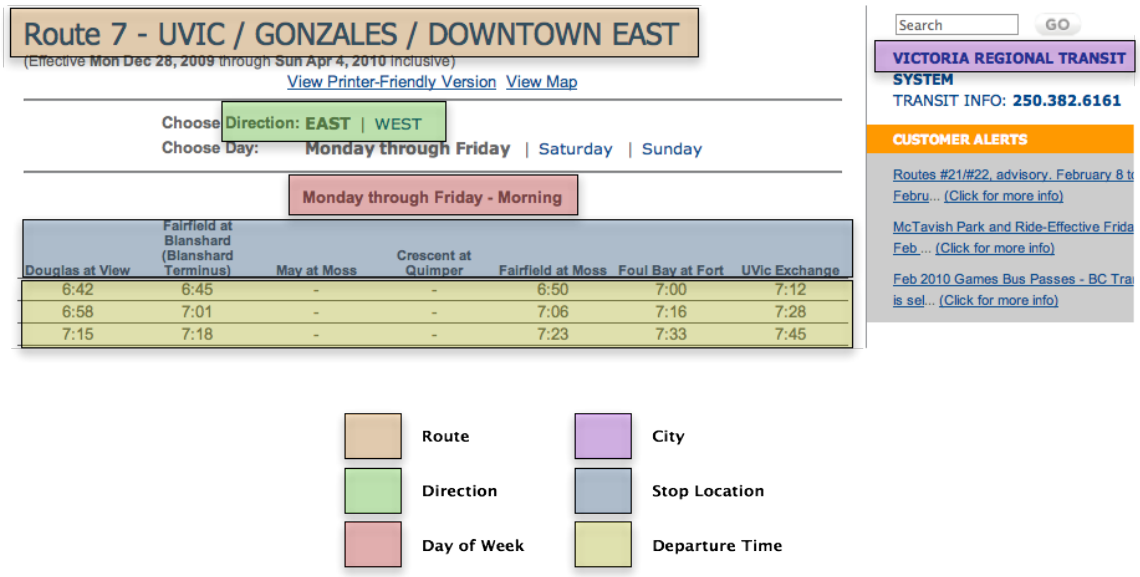

Figure 3.3: Various dimensions transit schedules leading to information bloat

3.1 Information Reduction

Many mobile devices have technologies that can assist in reducing the amount of effort required by the user when looking up transit departure times, including timing hardware, global positioning systems (GPS) and electronic compasses. Figure 3.4 outlines the process of looking up a departure time. The figure illustrates how these onboard technologies can assist in this lookup. First, the GPS can be used to narrow down the city, route and stop location. Second, an onboard electronic compass can detect orientation to determine the route direction. Finally, the internal clock can be used to select the current day as well as the next departure time.

There are two areas where user intervention is required. First, although GPS can be used to find the user's location, there is still the possibility of multiple transit routes at this location. For example a bus terminal sees many routes converge at a localized hub. In this case, the user would be required to select from a subset of routes. Second, routes usually have two directions. If the user is at a location where the route is not parallel to its label, or if the route was instead defined by its endpoints, she would need to explicitly describe her desired direction of travel.

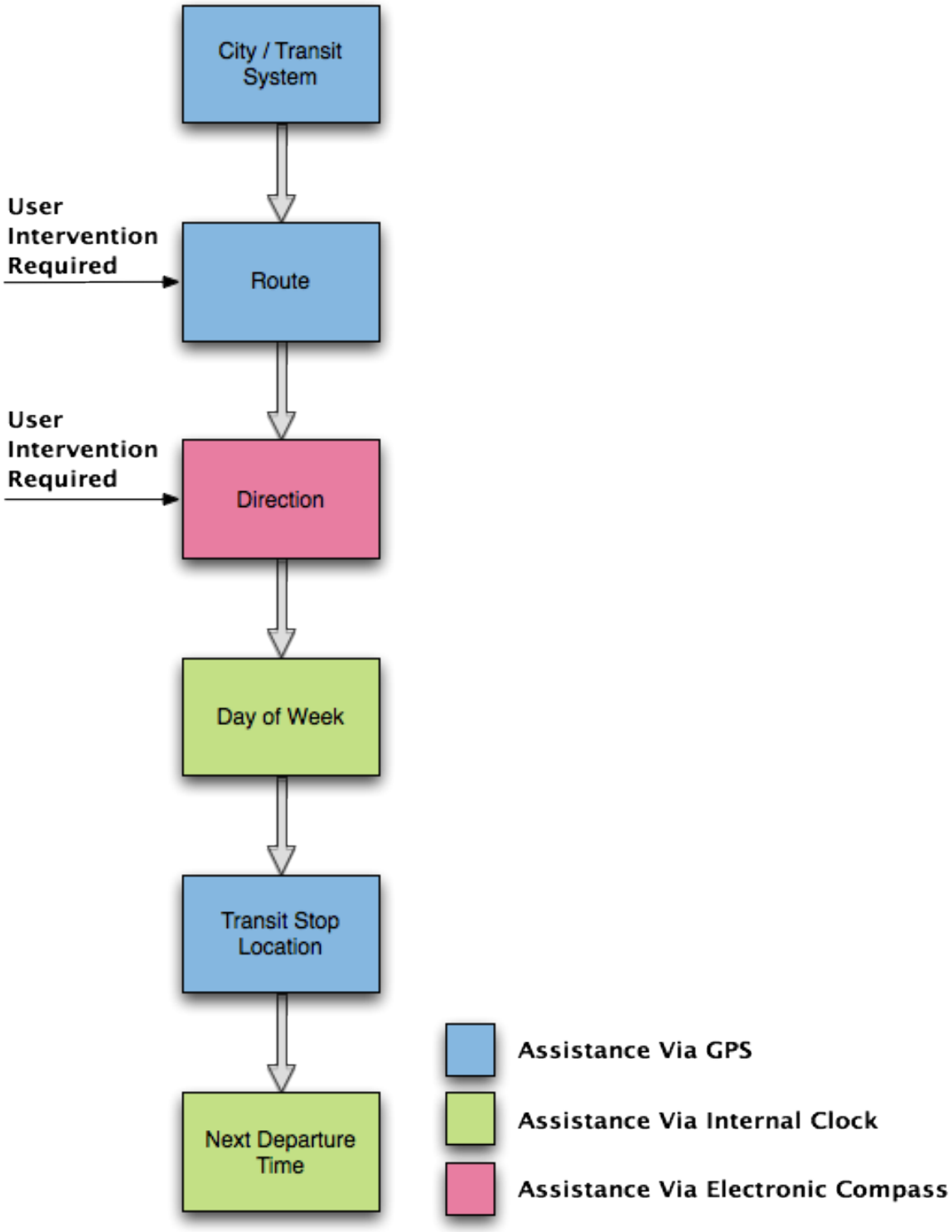

Figure 3.4: Transit information reduction process

3.2 Information Representation

A vast majority of Web applications adhere to the following architecture: The user begins by sending information to a Web server, usually by clicking on a link or filling out a form. The server uses this information, in the form of query parameters, to perform a query on a database. The data retrieved from the query is formatted in some way and sent back to the user to be consumed. Data is in its truest form within the database; however, high variability is introduced during the formatting process. Depending on the design, a high degree of coupling between information and markup used for presentation may be introduced. Figure 3.5 illustrates such an example.

```
<table class='scheduletable' border='0' cellpadding='0' cellspacing='0' width='100%'>
  <tbody>
    <tr>
      <td align='CENTER' colspan='14' class='css-sched-table-title'>
        <b>Monday through Friday - </b><b>Morning</b>
      </td>
    </tr>
    <tr valign='BOTTOM'>
      <td> </td>
      <td align='CENTER' width='100' class='css-sched-waypoints'>
        Douglas at View </td>
      <td> </td>
      <td align='CENTER' width='100' class='css-sched-waypoints'>
        Fairfield at Blanshard (Blanshard Terminus) </td>
      <td> </td>
      <td align='CENTER' width='100' class='css-sched-waypoints'>
        May at Moss</td>
    </tr>
    <tr align='CENTER'>
      <td class='css-sched-times'> </td>
      <td class='css-sched-times'>6:42</td>
      <td class='css-sched-times'> </td>
      <td class='css-sched-times'>6:45</td>
      <td class='css-sched-times'> </td>
      <td class='css-sched-times'>-</td>
    </tr>
  </tbody>
</table>
```

Figure 3.5: Legacy HTML source code for a transit schedule

The doctype of the document is HTML 4.01 Transitional. Accessibility software has difficulties with a such a document for the following reasons. First are the style-related attributes woven throughout the information. For example attributes such as `width` and `valign`, as well as deprecated tags like `<b>` and `<i>` belong in separate Cascading Style Sheets (CSS). Second is the use of HTML elements used for superficial document structure. Examples include using `<table>` tags and non-breaking spaces (`%nbsp;`) to help with the visual layout of other information. Finally, there is a lack of semantic meaning within the document. In many cases class attributes are used to represent the meaning of elements. For example 'css-sched-waypoints' are used to define the names of stop locations and 'css-

sched-times' are used for departure times. To an assistive screen reader or a screen scraping tool; however, these are just class attributes.

The argument in this scenario is not that such a presentation scheme is not effective. For many users, this is an acceptable solution. For users with disabilities or those using mobile devices, this form of information representation is too inflexible. The introduction to Chapter 3 described the use of an API to gain more direct access to information. A proposed structure for data is proposed in Figure 3.6.

```
<route name='UVic via Richmond' number='14'>
  <stop name='UVic Exchange' mapLetter='E' latitude='45.43454' longitude='-123.32342'
    <direction dir='west'>
      <day day='weekday'
        <time busId='123' bussType='Access' period='morning' time='0551' />
                <time busId='232' bussType='Access' period='morning' time='0632' />
      </day>
    </direction>
  </stop>
</route>
```

Figure 3.6: Proposed XML structure for transit data

3.3 Implementation and Deployment

Another driving force behind the mass adoption of mobile devices is the ever pervasive mobile Web. Increasing Wi-Fi hotspots and mobile 3G towers have made wireless broadband connectivity ubiquitous in many areas. For this reason, we chose to implement BUSSPASS as a Web service. Figure 3.7 shows the system's sequence diagram.

The task of finding a bus time begins with the user looking up transit routes near his current location. Onboard GPS is used to determine the user's current latitude and longitude coordinates, which are passed to the BUSSPASS client software. BUSSPASS uses this information to respond to the user with a list of routes that run near to where the user is currently located. Next, the user chooses the desired route and direction of travel. A Web request is then made to the BUSSPASS server software. The backend database uses the request information, as well as the current time of day to fetch the nearest departure times. These departure times are appropriately formatted and sent back to the client via the Web response. The user thus receives the next departure time for the route and direction he is interested in.

In the case where GPS is not available, we have designed an alternative approach using QR codes [29]. As shown in Figure 3.8, a QR code is a two-dimensional bar code capable of encoding significant amounts of data. By placing QR codes on bus-stop posts, a digital

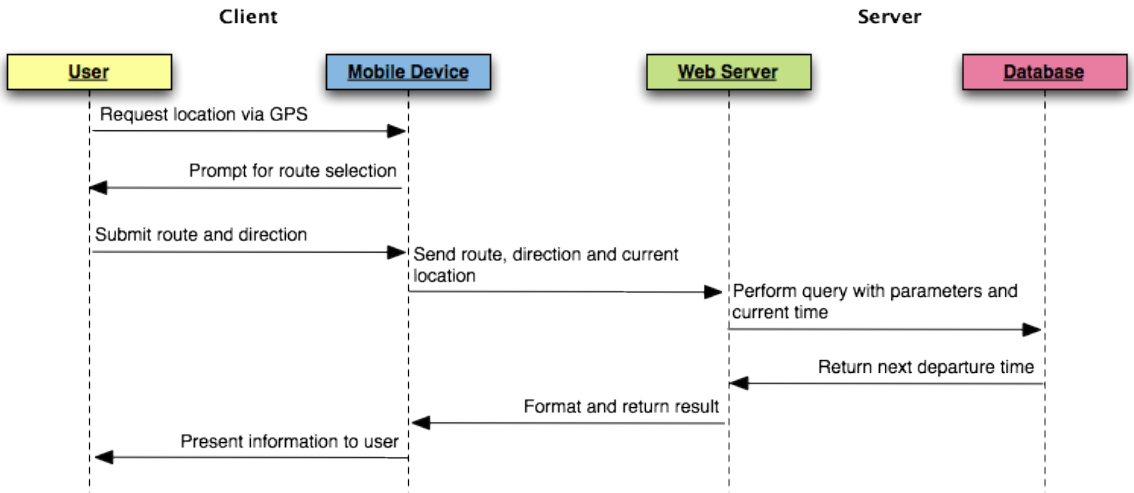

Figure 3.7: BUSSPASS sequence diagram

representation of that route's data could be made available. The user would approach the post and take a picture of the QR code with her mobile device's camera. Many mobile devices are equipped with software to decode QR codes. With the decoded route and direction information, the BUSSPASS client software would have the information it needs to proceed with departure time lookup.

Figure 3.8: QR code

The most significant problem with the QR code approach is getting support from transit authorities. QR codes that encode information at each stop would need to be installed - a costly process for which transit authorities may unwilling to pay.

Another design decision with significant tradeoffs is the choice of employing Web services instead of relying on local resources. Our implementation sees the majority of transit scheduling data on a remote server. The benefits of this approach include a significant re-

duction in the amount of storage required to hold transit data on the user's local device, flexibility to update the data repository when changes to the system occur (for example construction detours or holiday interruptions), and extensibility to add new transit systems to the repository. The main disadvantage is the required Internet connection.

Many users are less comfortable using the most current Web-based technologies. Further, several Web technologies are not accessible to the point where they can be used in the field. Traditional telephony-based interfaces provide a more familiar method to users. Figure 3.9 describes a telephone system based on the BUSSPASS data structures. Such a system would operate similarly to Google's GOOG411 service that uses telephony to interface to its automated backend services [30].

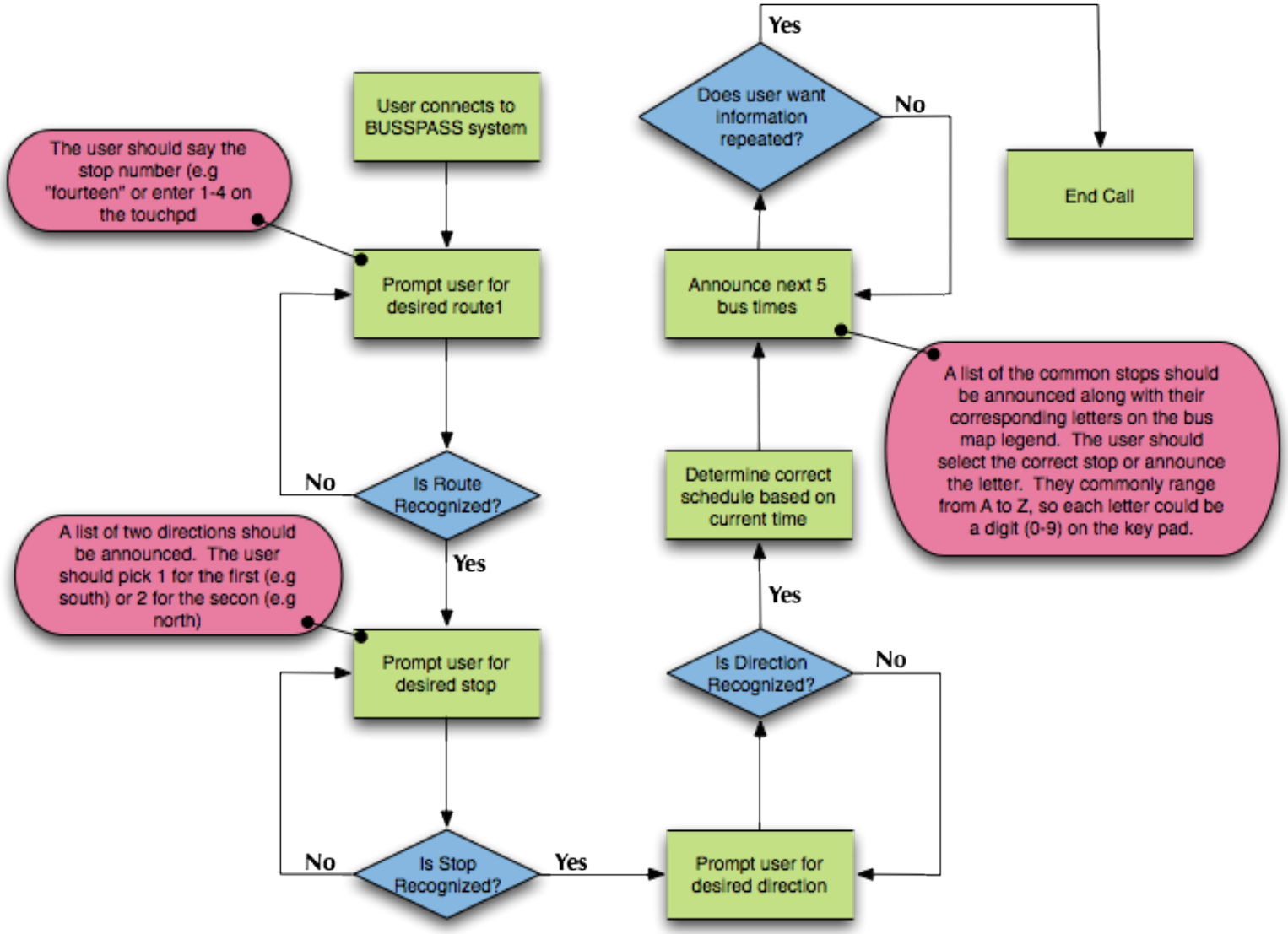

Figure 3.9: BUSSPASS telephone system

Chapter 4

Case Study II - Assistive Music Browsing

The de facto standard for presenting music information in media player software is to list the files' attributes in a tabular fashion, similar to that of the transit schedule (see Figure 3.2). Columns of this table represent various pieces of metadata about an audio file such as artist, title, genre, song length and many more. Rows represent the audio files in the music library. Apple iTunes[1] is an example of this type of user interface, and is shown in Figure 4.1.

This type of interface for mobile and accessible computing suffers the same sort of problem described in Chapter 3. If zoomed in on a particular subset of records, the user loses visibility of the table headings which provide context about the music library. Apple's iPhone OS employs a drill-down interface to address this issue (see Figure 4.2). The user begins navigating the collection at some high level (for example by artist or by album). Once a particular item is selected, the user is presented with a new table with more fine-grained information relative to it's parent table.

While this hierarchical approach to organizing music is an improvement, it is still highly textual and list-based. For example, a large music collection would see the user scrolling through large lists of artists, albums and songs. Having a screen reader read long lists of items soon becomes fatiguing. Another problem with this approach is the amount of effort and time required to maintain one's collection. Ensuring that audio files are correctly tagged and organized often requires the user to listen to each audio file, and make adjustments to the textual metadata. New tools are emerging that attempt to greatly simplify and automate the editing process, one such example being Pollux [31]. Pollux is an iTunes extension that automates the task of filling out metadata for the songs in one's music collec-

[1] Apple iTunes: http://apple.com/itunes

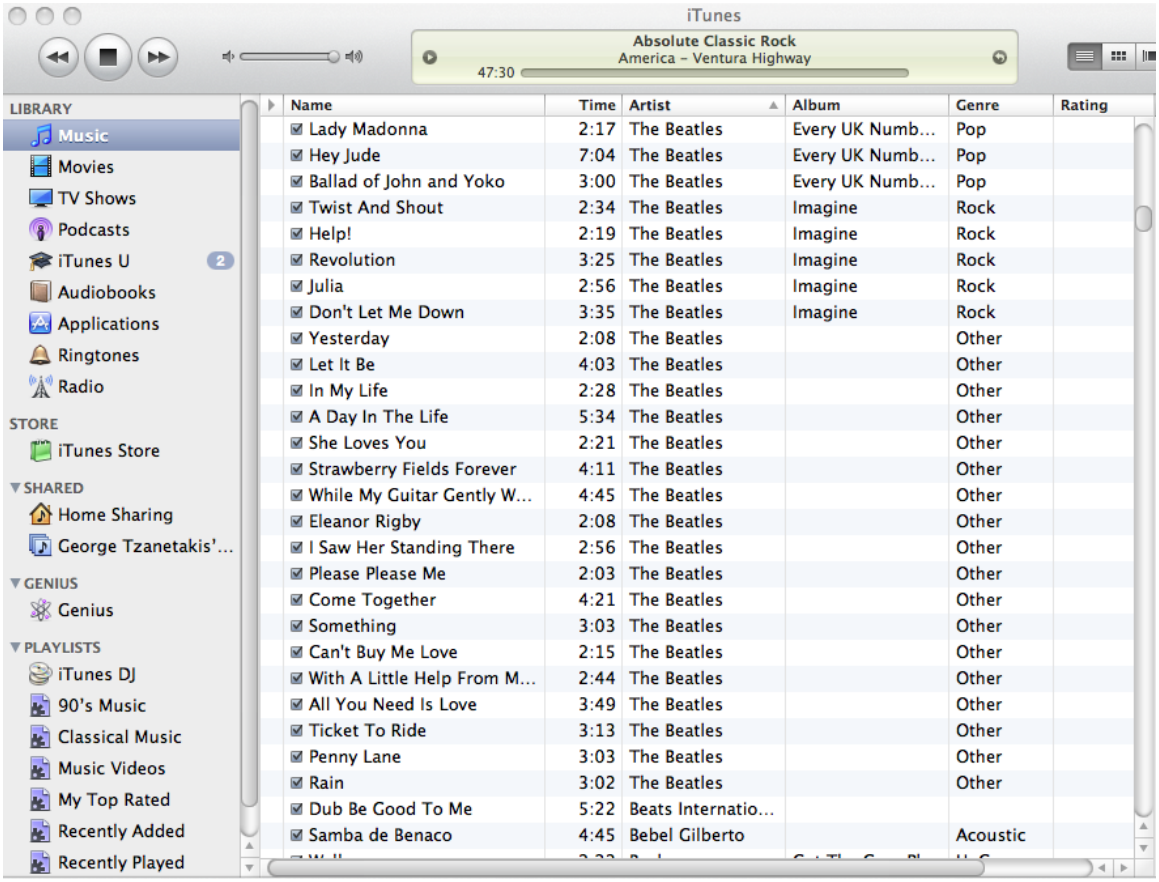

Figure 4.1: Apple iTunes

tion. First, the audio file is processed with a hashing function to compute a unique digital fingerprint of that file. The fingerprint is processed by the Amplifind service to identify the audio file [32]. Once the audio file is identified, services such as Last.fm [33] and MusicBrainz [34] are used to find additional metadata and artwork for that file.

With the acceptance of mp3's and large, inexpensive storage devices, personal digital music collections have grown to the size of thousands of songs. Without disciplined organizational habits, navigating one's collection can be a challenging task. The reduced screen size of mobile devices has proven to be a significant problem for presenting dense data such as this. Furthermore, users with vision or motor disabilities often find a text-based interface completely unusable.

Advancements in digital technology have made possible many novel means for classifying and organizing music including metadata, social characteristics (for example mood), and acoustic similarity [35]. With new search techniques, such as query-by-example, users

Figure 4.2: iPhone music application

are able to find unexpected yet acceptable results [36, 37]. Many prototypes for visualizing large music collections have been developed. Islands of music sees clusters of closely related audio files shown as islands, using self-organizing maps and low level features [38]. Artist map presents a music collection as a two-dimensional map based on social and acoustic features, and allows the user to create playlists by drawing paths on the map [39]. Musicream allows the user to build playlists by selecting particular songs which automatically attract similar songs [40]. With Torrens's system, users work with one of three visualizations (tree-map, disk or rectangle) by slicing and rearranging subsections to create playlists [41].

4.1 MarGrid Mobile

To address these concerns, we have developed a novel music navigation program for mobile devices [42]. MarGrid Mobile, as shown in Figure 4.3 is a touch-based, mobile extension to the desktop MarGrid program. Both pieces of software rely on the feature extraction of audio files and self-organizing maps (SOM) for presentation. Marsyas is an audio processing framework used for these tasks [43]. The software works as follows:

First, feature extraction is applied to a collection of audio files. This generates a set of descriptive, high-dimension feature vectors - one for each audio file. These features include the Spectral Centroid, Rolloff, Flux and Mel-Frequency Cepstral Coefficients, among others. The audio file is sampled throughout a 30 second interval, and each feature is determined by maintaining a running average and standard deviation. Finally, the feature vector values are normalized within a range of 0 and 1 inclusive. Second, a training phase sees a subset of feature vectors used to construct an SOM. The SOM is initialized with random

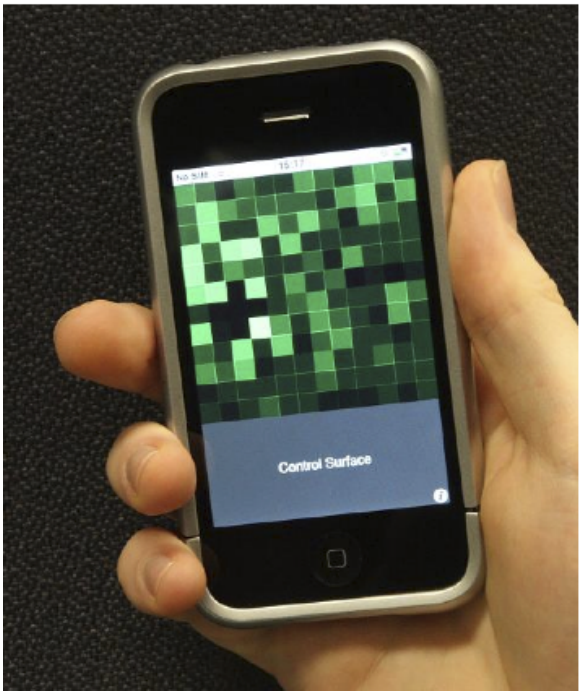

Figure 4.3: MarGrid Mobile

values. The Euclidean distance of each vector is calculated and applied to the SOM. Once the most appropriate SOM node for the sample vector has been determined, neighbouring nodes reorganize themselves to accommodate the new data. After iterating through each of the training vectors, the SOM is defined and ready to accept the remaining audio file vectors of the collection. The remaining vectors are added simply by finding the SOM nodes with the best matching vectors [42].

Figure 4.3 shows the mobile implementation of this software. The SOM is displayed as a two-dimensional grid, where each square of the grid represents a SOM node. The colour intensity of each node represents the relative number of audio files associated with that node; a light colour indicates more audio files than a dark colour. Because music genres are clustered around different areas of the grid, the user is able to navigate her collection by touch rather than by sight. For example, rock music may be located in the lower left portion of the grid, while jazz may occupy the upper left area. By touching the top-left square of the grid, the user would hear a jazz recording. By sliding her finger down the left edge of the grid, the music would transition into rock. The lower third of the touch area is reserved for other controlling gestures. For example a right swipe skips to the next song in that grid square, a left swipe plays the previous song, and a top pauses and resumes playback.

As mentioned in Section 2.3.1, mobile device resources are considerably more lacking

compared to their desktop counterparts. Also, significant restrictions are placed on applications that target the iPhone platform, including limited filesystem access, single-process execution, and isolated program communication. For these reasons, we chose to implement MarGrid Mobile as an application that works in tandem with the desktop version. Specifically, Marsyas was used on the desktop for feature extraction and the generation of the self-organizing map. A text file that describes this map was then uploaded to the iPhone, along with the user's music collection. The iPhone application used the text file to recreate the self-organizing map. The major disadvantage of this approach was, due the the lack of filesystem access, the duplication of audio files on the user's device. For example, if the user had a collection of songs synced to the iPhone through iTunes, these files were not available to be accessed from our application. Instead, a new playlist was created, and the songs it contained were uploaded to the iPhone as part of our application bundle, regardless of whether or not they already existed. We point out that this limitation is inherent to the architecture of the iPhone platform, not our application specifically.

Also, as part of this project, we were able to port most of the Marsyas framework to the iPhone platform. The one exception was Marsyas's audio playback functionality. This functionality relies on Core Audio, a Mac-specific framework, which is not available to the same capacity on the iPhone [44].

4.2 User Feedback

A difficult aspect of HCI is the validation and evaluation of results. Section 2.1 outlined several factors that contribute to good UI design; however, these factors can be challenging to quantitatively measure. Although a formal user study was not conducted, we have collected basic input from individuals with disabilities who have an invested stake in such projects. This feedback was primarily used as part of an iterative approach to developing the three prototypes described in Sections 3, 4 and 5.

To evaluate the usefulness of the assistive music browsing software, a visually disabled user was interviewed. This included not only mobile assistive browsing using MarGrid Mobile (iPhone), but also desktop software using both a mouse-based and keyboard-based implementation for navigation. Using a collection 1000 songs of varying genres, he was asked to perform two tasks:

- Find a song from a specific genre (for example, rock, blues or jazz).
- Find a song that sounded similar to a song that was played by the researcher.

Based on impressions from performing these tasks multiple times, the user completed the following questionnaire:

- How easy was it to navigate the music collection looking for a specific genre?
- How easy was it to navigate the music collection looking for a similar song?
- Comment on the difference of your performance among iterations of the two tasks.
- Would you want to use this system on a regular basis?
- Other comments.

User feedback indicated that tactile and auditory information was used more than visual information. On the desktop software, the user would orient himself with the grid by starting in one of the corners. The technique is well-known and employed by popular UI's such as Apple's Mac OS. The menu is at the top of the screen and the user can get to it easily by moving the mouse to the top of the screen. Instead of trying to place the mouse on a specific target of the screen, the bounds of the display act as a stop so the user cannot "overshoot" the target. The same approach was used with the music browsing software. Instead of trying to place the mouse on a particular grid square target, the user began by moving the mouse to the top-left corner. The application restricted the mouse to within its bounds which made overshooting the corner target impossible. Starting at a particular corner, the user navigated the collection by moving to adjacent grid squares and listening to the samples of music they contained. The most difficult task was the initial learning of the genre mappings to the grid. Each training phase sees genres placed at unpredictable areas of the grid. The user must therefore learn these areas to be more efficient when searching and navigating. The user found it simple to locate a specific genre, and relatively simple to locate similar sounding songs. A similar user approach was taken on the iPhone. By using the edges of the device, the user could easily find the corners and bounds of the grid, and navigate the collection from there. He concluded that the touch-based iPhone was more efficient than the mouse-based implementation because the entire bounds of grid could be interpreted (though touch) and navigated more quickly. Further, because he was accustomed to using iTunes with dedicated accessibility software, he felt that traditional desktop music software was more efficient. However, because many mobile platform do not provide accessibility software, this approach may be more viable.

Chapter 5

Case Study III - The Implicit Patching Model

Many computer applications are dedicated to the presentation and configuration of domain specific information through the use of schematics and block diagrams. This model of information presentation is pervasive throughout many disciplines; Science, business, education and productivity are but a few examples.

5.1 Explicit Patching

Regarding the flow of information, a common visualization is to consider information endpoints as nodes. The information thus flows from source node, through internal nodes, to a sink node. This is shown in Figure 5.1

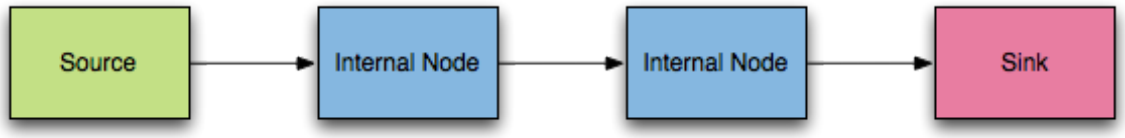

Figure 5.1: Information flow

This model is known as *Explicit Patching*. Source nodes have output ports, sink nodes have input ports, and internal nodes have both output and input ports. A network of nodes is created by explicitly establishing connections between the input port of one node and the output port of another.

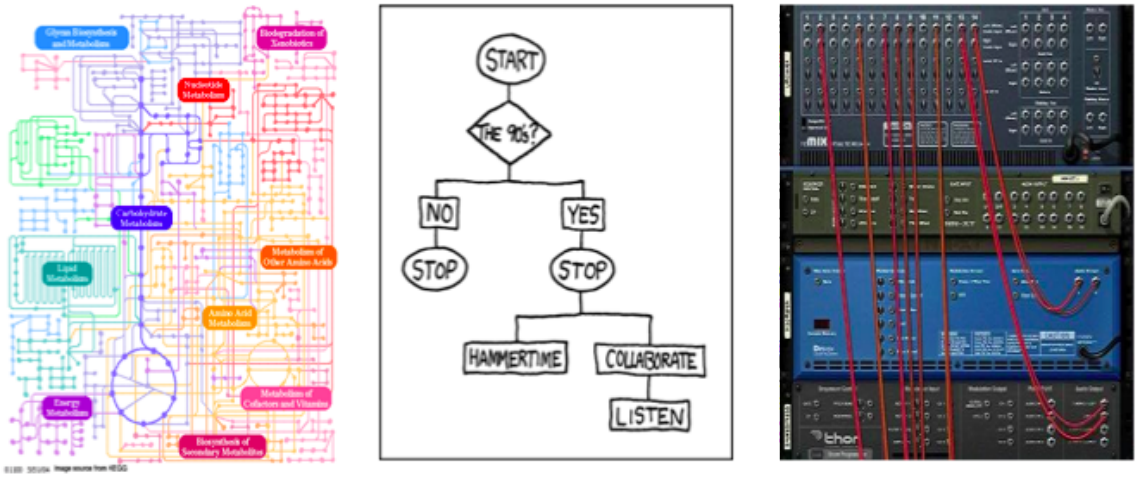

Figure 5.2: Examples of information flow

As shown in Figure 5.2 the concept of representing information networks through the use of block diagrams, schematics and flowcharts is employed in many disciplines. In biochemistry metabolic charts describe the flow and alteration of substrates as they are transformed through enzymatic reactions. Mind mapping is a common approach to organizing the flow of information in productivity and business domains. In engineering, the transformation of signals from source to sink are commonly expressed in this manner. The term "patch" comes from the audio industry where audio processing components were patched together with physical wires.

The explicit patching model is a common choice for representing basic information networks because it is simple to use and it describes the network in a clean, straight-forward fashion. However, for non-trivial networks, this model becomes problematic for at least three reasons. First, as the number of nodes in a completely connected network increase, the amount of explicit connections grows geometrically. This causes the network visualization to become cluttered and unwieldy. Second, in a many-node, highly coupled network, manual creation and editing of explicit patches is time consuming for the user. Finally, implementing behaviour to maintain explicit patches as nodes are rearranged can be complex depending on the chosen drawing engine.

As an example, consider the following scenario: The network in Figure 5.3 shows five internal nodes receiving information from a single source node. The explicit patching model requires the user to create five patches. Thus far the network is manageable;

	Sink=1	**Sink=2**	**Sink=3**	**Sink=4**	**Sink=5**
Sink=1	1	2	3	4	5
Sink=2	2	4	6	8	10
Sink=3	3	6	9	12	15
Sink=4	4	8	12	16	20
Sink=5	5	10	15	20	25

Table 5.1: Explicit connections required to completely connect source and sink nodes

however, consider the addition of a second source node whose information is also required by each of the internal nodes, as shown in Figure 5.4. The addition of one node results in an increase of five explicit patches (an increase of 2x in this case). Manually adding these connections to the network is time consuming, and the network is significantly more cluttered.

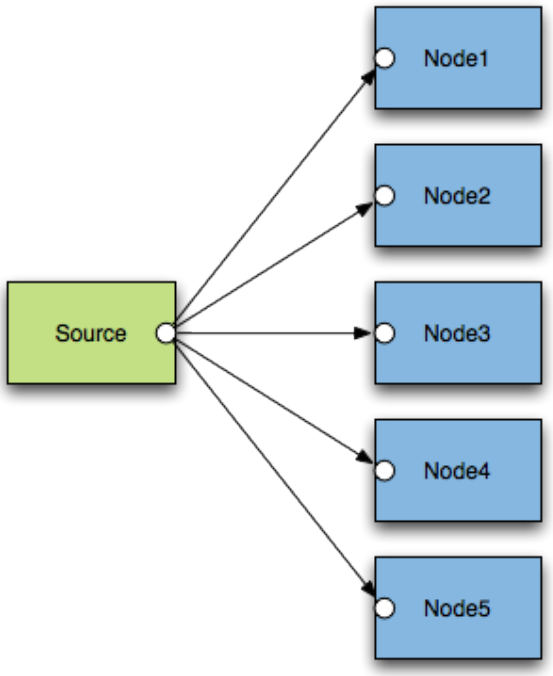

Figure 5.3: Explicit connections between one source node and five sink nodes

Table 5.1 shows the number of explicit patches required to connect a complete network of source nodes to sink nodes. In other words, if there were x source nodes and y sink nodes, each y node would receive input from every x node.

In a best-case situation, the manual addition of explicit patches is tedious for the user,

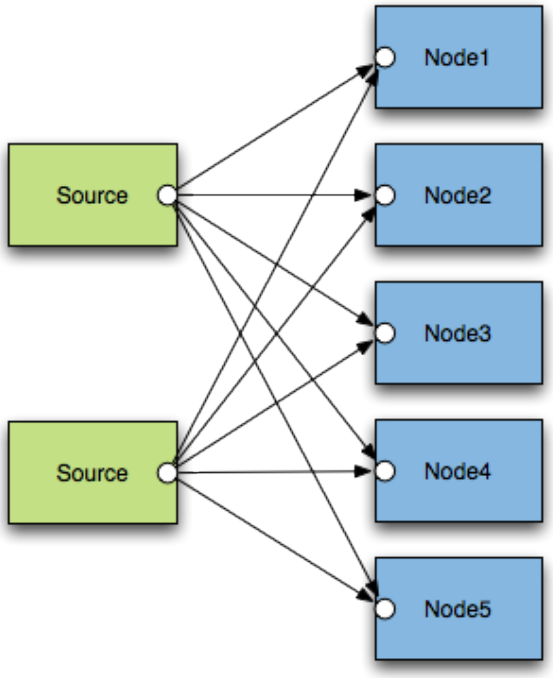

Figure 5.4: Explicit connections between two source nodes and five sink nodes

however there exist use cases whereby the explicit patching model is completely unusable: First, as mobile computing trends increase, more users rely on devices such as the iPhone to be productive. The explicit patching model is a poor fit for the reduced screen real estate of such devices. Second, millions of users have some form of disability that affects the way in which they interact with a computer. The nature of the explicit patching model requires the user to make explicit connections between nodes, usually by clicking and dragging from the output of one node to the input of another. For those with physical or visual disabilities, the click-and-drag paradigm is not a feasible solution. Finally, many types of information networks have some form of hierarchical structure associated with them. In the explicit patching model, nested nodes require additional notation in the diagram to represent them, again contributing to disorder within the document.

5.2 Implicit Patching

To alleviate the overhead of creating explicit connections, and to create cleaner visual representations, an *Implicit Patching* model is proposed. Unlike explicit patching, an implicit patching model conveys the connections among nodes based on their proximity to one an-

other within the network. This is shown in Figure 5.5. This example shows the same basic network as in Figure 5.1, except an implicit patching model is used instead of the explicit patching model. From this diagram in can be inferred that some signal starts from the source, and flows in the right-handed direction until it reaches the sink. With this model, users need not concern themselves with manually connecting nodes. Instead, the patching is performed implicitly based on the order in which the nodes are added to the network. In fact, the operation set required to create a network using the implicit model is very basic. It consists of two operations: `createNode()` which creates a new node within the document, and `appendNode()` which appends a newly created node to another pre-existing node.

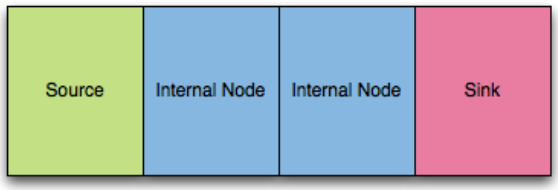

Figure 5.5: Implicit patching

To facilitate the creation of implicitly-patched networks, a set of specialized nodes are defined. For example, the network shown in Figure 5.1 can be recreated with the implicit patching model through the use of *composites*. A composite can be thought of simply as a black box. It has inputs and outputs just like any other node, however the user need not concern himself with the inner workings of the box. In general, a composite can consist of any arbitrary sub-network created by the user. There are, however, a few composites that play a more specific role within a network. These composites may be thought of as operators to the network. A *series* is the most basic composite operator. As shown in Figure 5.6 nodes appended to a series are connected in a linear fashion. A *fanout* (shown in Figure 5.7) is another important network operator; it routes its input to each of its appended nodes. Typically the arrows in this figure would not be part of the visualization. They are included here to show how information is moved through the network.

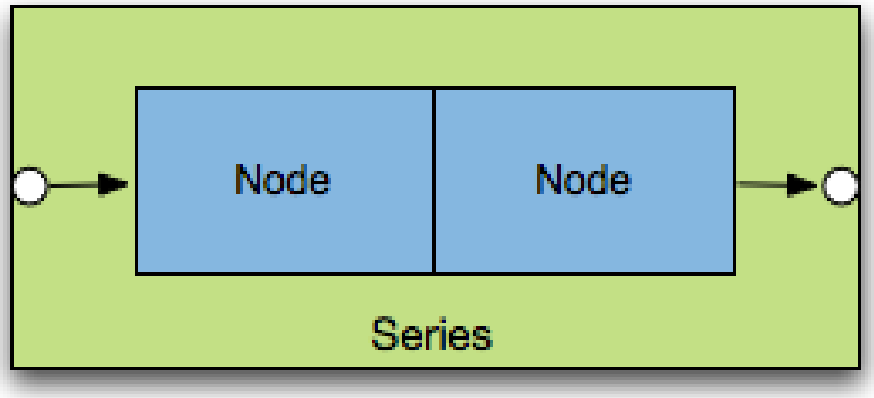

Figure 5.6: A series component

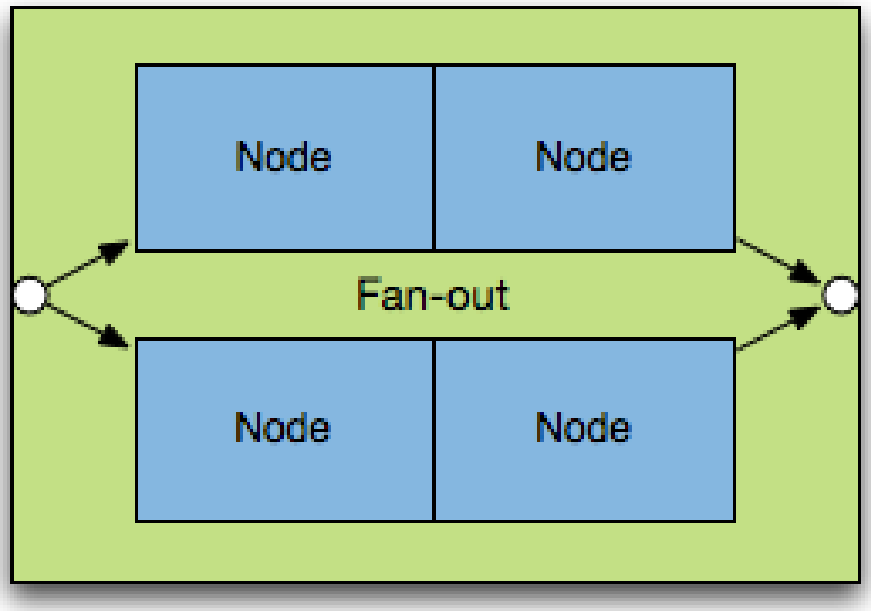

Figure 5.7: A fanout component

Consider again the scenario described in the Figure 5.4. There are 5 internal nodes that wish to receive information from two source nodes. This network can be created using the series and fanout composites described previously. Pseudo-code might look similar to that shown in Figures 5.8 and 5.9.

```
// Create the composites
network = create("series")
fan1 = create("fanout")
fan2 = create("fanout")

// Establish network internals by appending leaf nodes
fan1.append(source1)
fan1.append(source2)

fan2.append(innerNode1)
fan2.append(innerNode2)
fan2.append(innerNode3)
fan2.append(innerNode4)
fan2.append(innerNode5)

// Complete the network by linking the composites:
network.append(fan1)
network.append(fan2)
```

Figure 5.8: Implicit patching pseudocode.

```
// Create the composites
network = create("series")
fan1 = create("fanout")
fan2 = create("fanout")

// Establish network internals by appending leaf nodes
fan1.append(source1, source2)
fan2.append(innerNode1, innerNode2, innerNode3, innerNode4, innerNode5)

// Complete the network by linking the composites:
network.append(fan1, fan2)
```

Figure 5.9: Simplified implicit patching pseudocode.

A visualization of this network is shown in Figure 5.11. As a comparison, the pseudocode required to build the same network with explicit patching would look similar to that shown in Figure 5.10. Without the conventions present with implicit patching, information must be repeated many times to create explicit connections among nodes.

```
connect(source1.out, innerNode1.in)
connect(source1.out, innerNode2.in)
connect(source1.out, innerNode3.in)
connect(source1.out, innerNode4.in)
connect(source1.out, innerNode5.in)

connect(source2.out, innerNode1.in)
connect(source2.out, innerNode2.in)
connect(source2.out, innerNode3.in)
connect(source2.out, innerNode4.in)
connect(source2.out, innerNode5.in)
```

Figure 5.10: Explicit patching pseudocode

The implicit patching model has numerous advantages, some which impact accessibility directly. First, visualizations of networks are cleaner without dozens of patch connec-

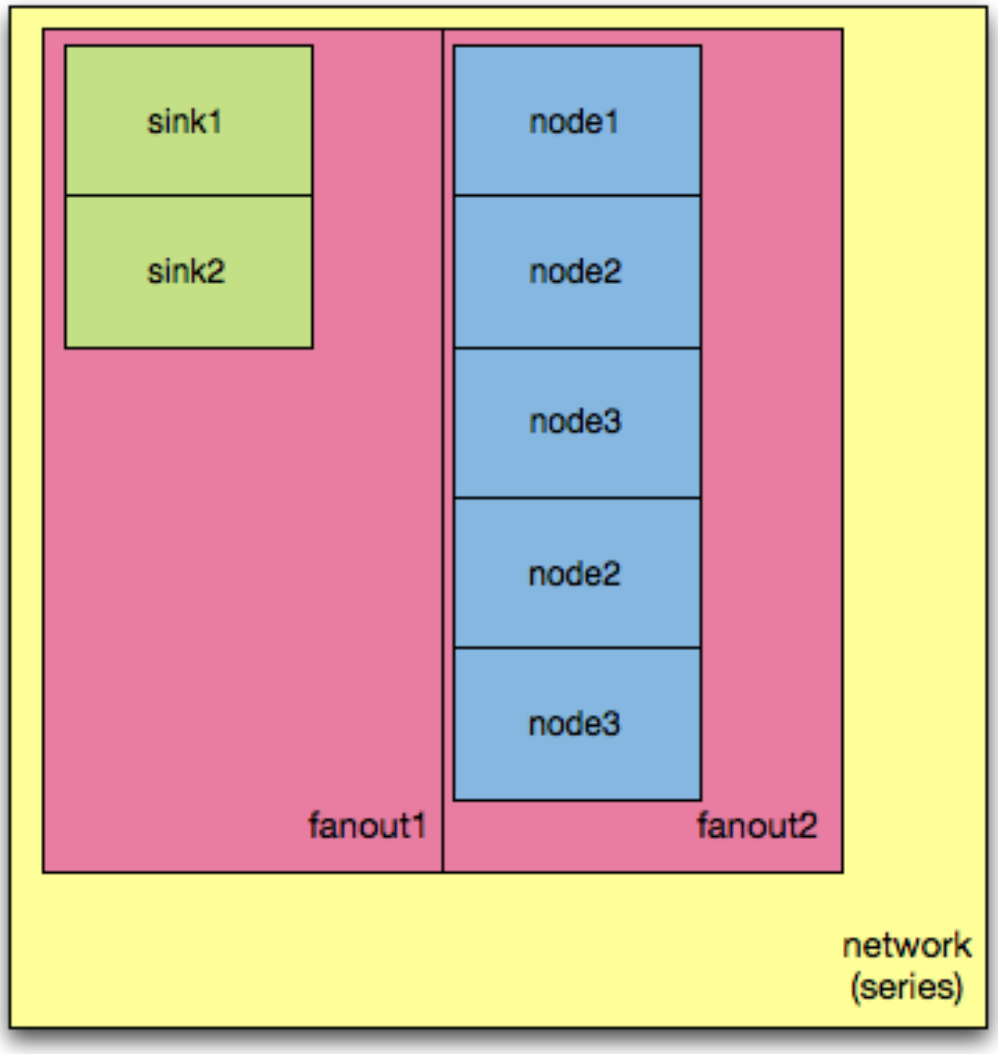

Figure 5.11: Implicit patching network visualization

tions hindering the view of more important objects. The addition of nodes or connections forces the user to reorganize the figure to accommodate the new objects - a time consuming process. Second, hierarchies of network internals are supported and facilitated through the black box model of the composite operators. Third, users (especially those with disabilities) need not rely on the click-and-drag paradigm to create a network. Instead, effective use of keyboard-mapped creation and appending functions can be used to build a network from the inside out. Fourth, composites also encourage re-usability. Networks can be created and packaged up as a black-box composite, which in turn can be reused in other networks. Finally, implementation is simplified. The composite design pattern treats leaf nodes the

same as composite nodes. Placement of components is standardized, as this visualization is the intent of the model. In other words, drawing code need not be implemented to track individual nodes and the explicit patches that connect them.

5.3 Scenario: Audio Playback with Explicit Patching and Implicit Patching

This section serves to compare and contrast explicit and implicit patching by illustrating a mockup implementation of a simple music playback network. Although this example employs ideas specific to the Marsyas audio processing framework (as described in Section 4.1), the underlying concepts regarding information flow visualization and user interaction can be extrapolated to any specific domain. The mockup is based on the Apple iPad, and it is expected that the user would be interacting with the application through a touch interface. Both versions of the mockup utilize three nodes to achieve playback of an audio file: an input node, a gain node and an output node. The job of the input node is to locate and load an audio source which in this case is a file named `mySong.mp3`. The input node then passes this information to an internal gain node. The gain node serves to adjust the volume of the audio passed to it. Finally the audio information makes its way to an output node. The output node is responsible for routing information to its destination. In this example the destination would be audio speakers.

In the explicit patching environment, the user must first create the nodes required to complete the audio playback task. Figure 5.12 shows the application canvas with three newly created nodes on it. At this point the three nodes are independent units whose interaction must be initialized explicitly by the user.

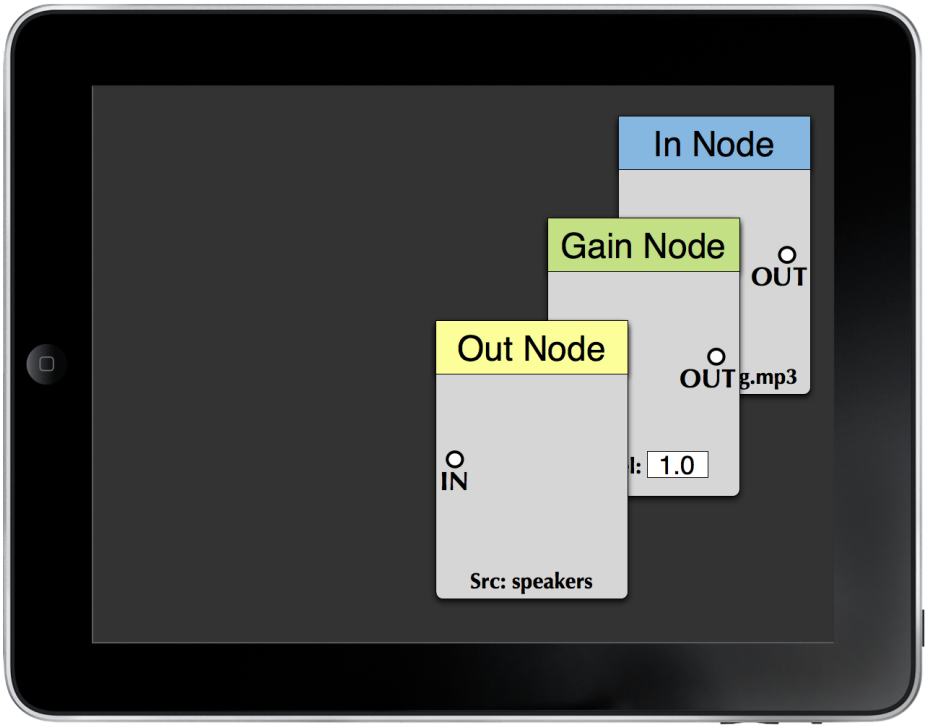

Figure 5.12: Audio playback scenario - explicit patching: Three newly created nodes

Once the nodes are created, the user must organize them into some coherent structure. This is done by touching a node and dragging it to its new location. Figure 5.13 shows the nodes organized into an apparent left-to-right structure, however there is still no interaction established between them.

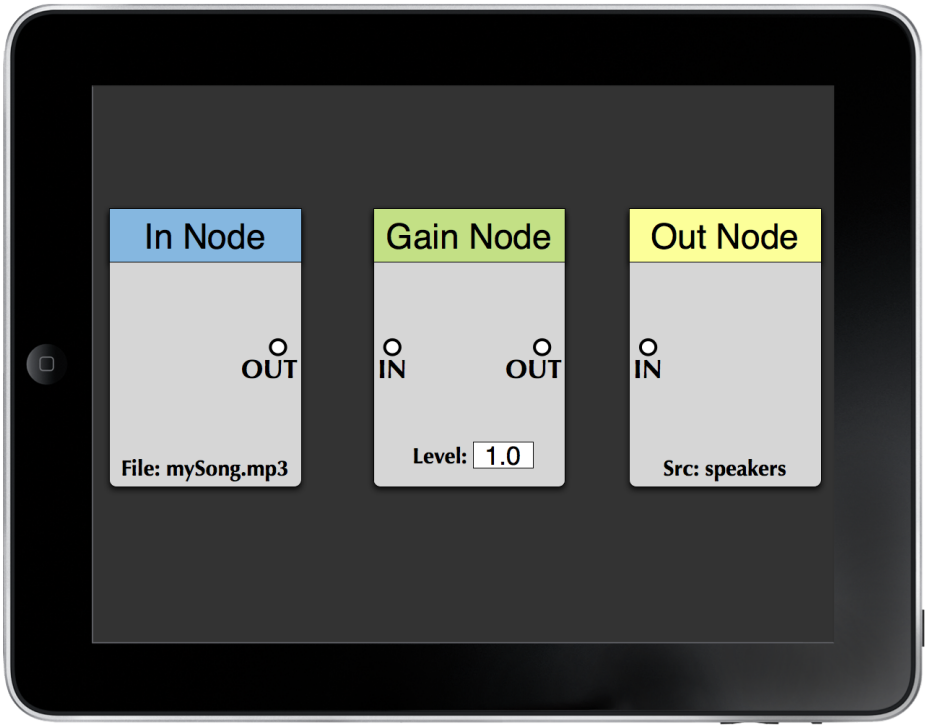

Figure 5.13: Audio playback scenario - explicit patching: Nodes organized from left to right

Finally, to establish the network, the user must initialize connections among the nodes. This is achieved by dragging a patch from the output of one node to the input of another. This process is illustrated in Figures 5.14 and 5.15. Note that in this example, the nodes are organized in a simple linear structure; however, the structure of more sophisticated networks would be much more complicated and may require many connections among nodes.

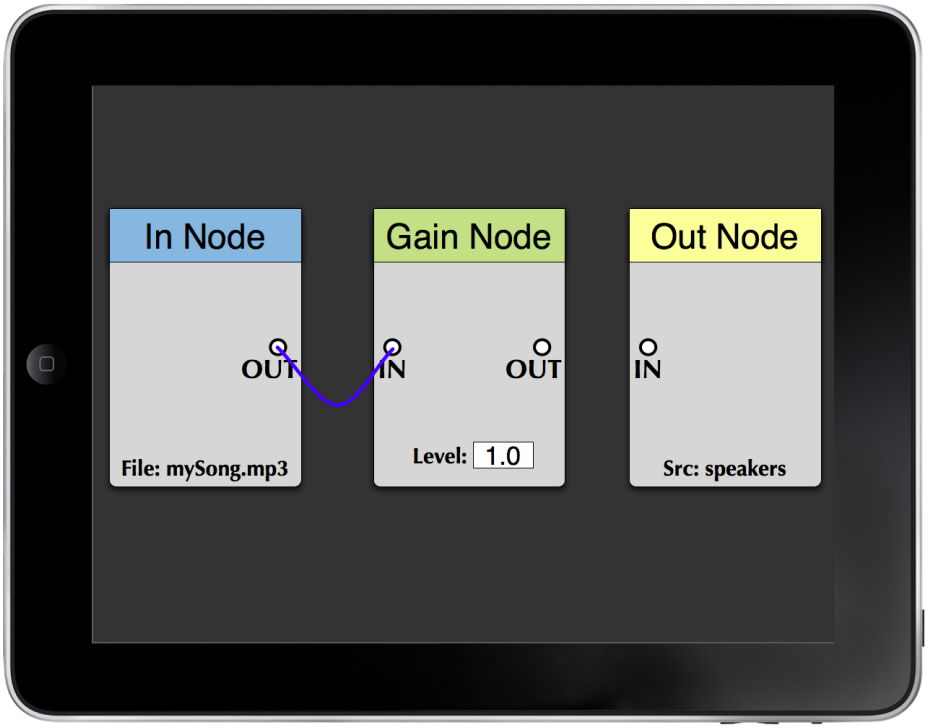

Figure 5.14: Audio playback scenario - explicit patching: A partially connected network

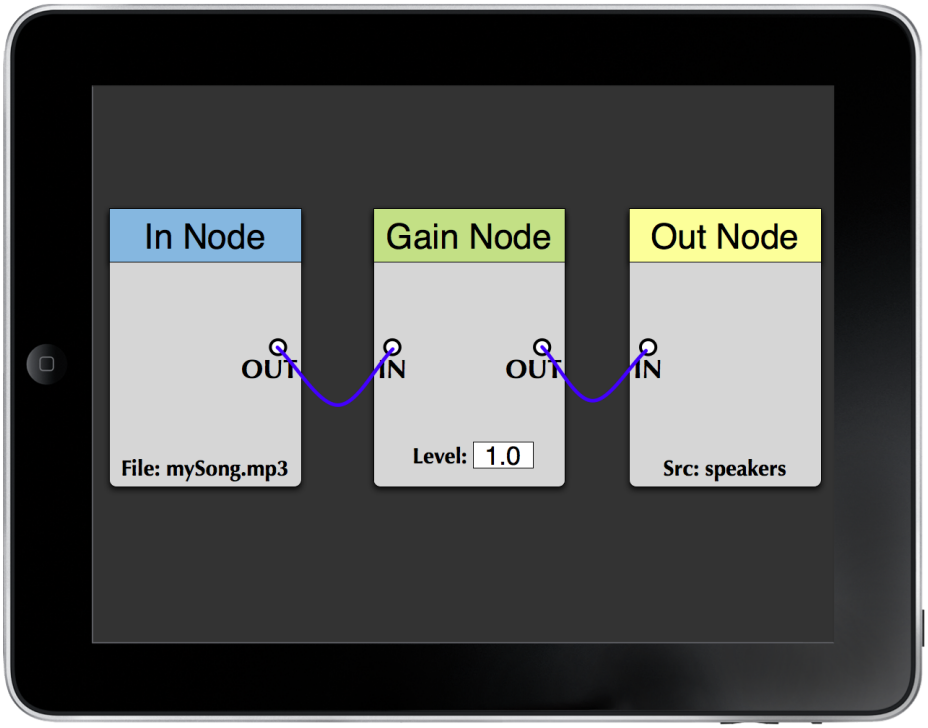

Figure 5.15: Audio playback scenario - explicit patching: A complete network

Conversely, the implicit patching environment acts as follows: By convention, all implicit patching networks consist of a single series component at its outermost level. To build out the network, nodes are appended to the internal of the single parent node. A blank canvas consisting of a single series component is shown in Figure 5.16.

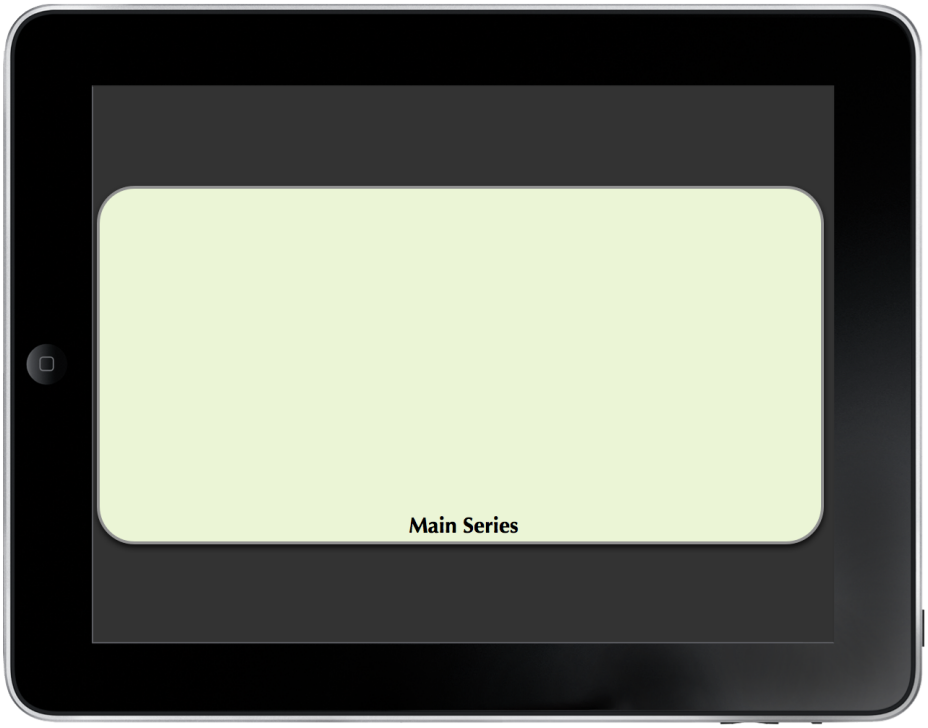

Figure 5.16: Audio playback scenario - implicit patching: A blank canvas with a single series component

The user then builds out the network by appending the input node (Figure 5.17), gain node (Figure 5.18) and output node (Figure 5.19) in that specific order.

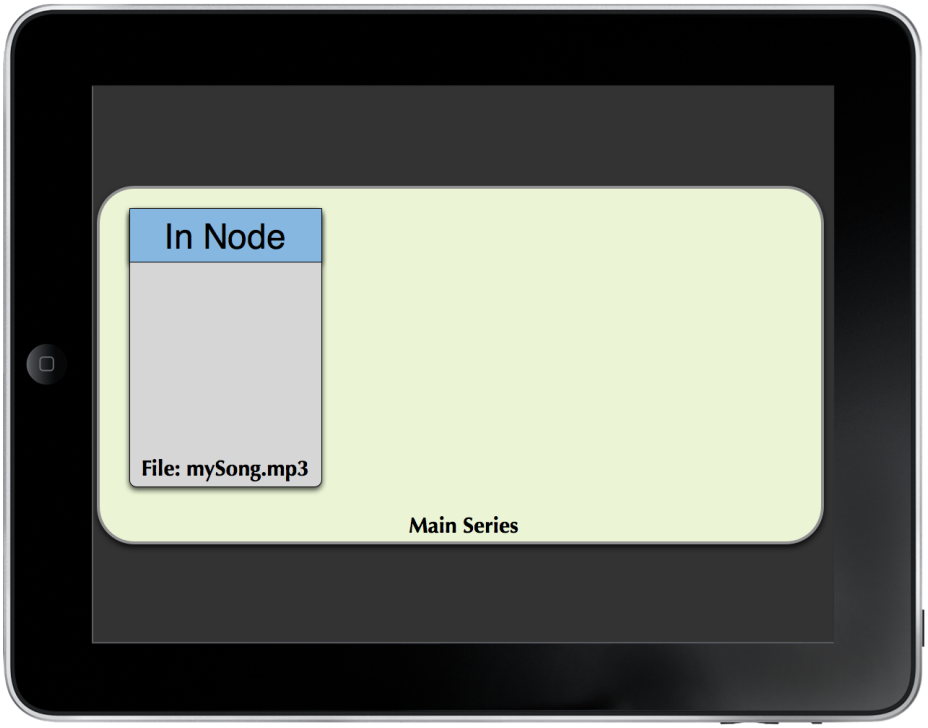

Figure 5.17: Audio playback scenario - implicit patching: Addition of the input node

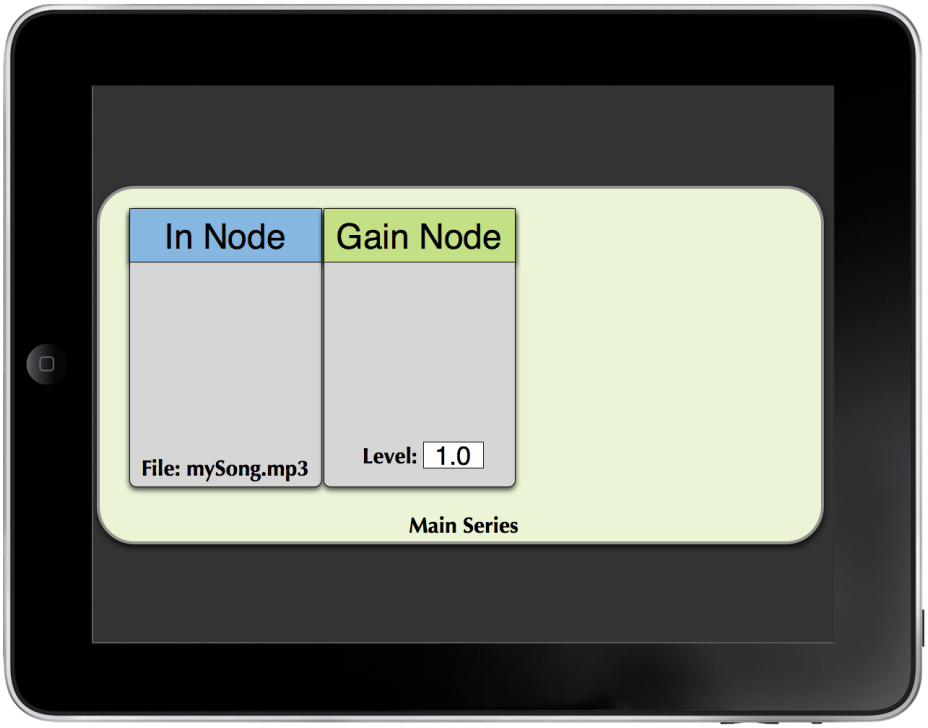

Figure 5.18: Audio playback scenario - implicit patching: Addition of the gain node

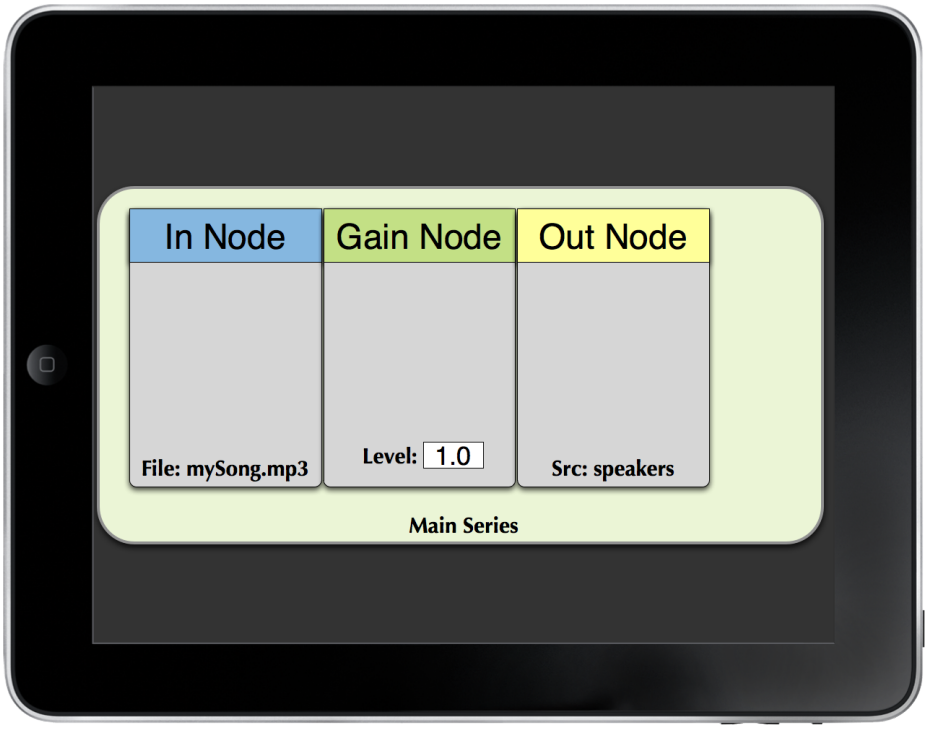

Figure 5.19: Audio playback scenario - implicit patching: Addition of the output node

It can be seen that the implicit patching environment has several advantages regarding accessibility. First is the avoidance of having to manually position elements on the canvas. The implicit model employs a structure that is more consistent and predictable. This consistency is advantageous for blind users who need not search around the canvas for the node they are interested in. Additionally users with motor skills limitations are not burdened with manually positioning elements on the canvas. Implicit patching also promotes the idea of nesting. Any of the three internal nodes in aforementioned figures could themselves be composite nodes which have their own internal nodes. By choosing an appropriate level of abstraction, the user can focus on the area of the network he is interested, resulting in a clean, uncluttered interface. Finally, by appending nodes in a specific order, the user interface need not maintain or present an explicit set of connection lines among nodes. For highly connected networks, this vastly simplifies the user interface and saves the user time when making connections and repositioning nodes.

Chapter 6

Case Study IV - WIKSSI: Accessible Language Education

The Web provides a truly dynamic representation of knowledge. Web 2.0 facilitates interactive information sharing, heightening expectations for collaborative learning. Likewise, the vision of the Semantic Web (Web 3.0) to support navigation based on relationships between entities and their properties has potential to leverage deeper and emerging organizational structure within a knowledge base. In terms of evolving this knowledge base, even from within developing regions, ubiquitous technology increasingly allows more of the worlds population to collect and output data in many forms.

Unfortunately, current Web applications leveraging existing infrastructure to support knowledge management for the purpose of education are still falling short of this global initiative. For example, support for language education, heritage preservation, alternative educational environments, scientific studies and disaster relief are all still largely static and often unidirectional.

The traditional WIKI (What I Know Is), made popular to the masses by WIKIpedia [45], supports anonymous content contribution shown to be successfully monitored by contributors to ensure accuracy. Course Management Systems (CMS) such as Moodle [46], Blackboard [47] or Connex [48] provide existing infrastructure for educational content management, conventional forum style support and some online chat functionality. FirstVoices [49] and the Virtual Learning Lodge (VLL) [50] provide solutions that are much more tailored to the specific needs of the Aboriginal learner. FirstVoices focuses on language education motivated by demands for Aboriginal language revitalization while the VLL, drawing more from social networking in its design, supports a wider spectrum of educational content.

Specifically in the domain of Aboriginal language revitalization, education that is in keeping with the culture and puts a high emphasis on social interactions and strong relationships to people and places is essential. Studies have shown that existing infrastructure is falling short of this social requirement in a population that has limited exposure to and experience with technology [51].

Social networking sites such as Facebook [52] and Second Life [53] provide a more interactive and community-based environment but they lack the ability to organize educational information in an accessible format.

Web 2.0 applications have made possible new tools and techniques for pedagogy [54, 55]. It has also greatly facilitated and encouraged community-driven content generation. *Crowdsourcing* has become an essential tool for problem solving [56, 57] in a variety of domains, including gaming [58], education [59, 60], news [61] and disaster relief [62, 63]. Content relevance is established both by a democratic voting process and the reputation of the user from whom the content was submitted. Semantic relationships among content is created though a *folksonomy* of user-submitted tags [64]. Web services provide the means for users with mobile devices to participate as well [65].

As the Semantic Web begins to take hold, opaque technologies such as Flash are being phased out in favour of standards-compliant alternatives. At the forefront is HTML 5 [66]. With its new document structure and multimedia tags, HTML 5 will significantly improve Web accessibility for disabled users.

On the mobile front, significant improvements in both hardware and software have made adoption of this technology more feasible for people with disabilities. The Android and iPhone platforms provide screen reading and / or screen magnifying capabilities [67, 68].

6.1 Developing the WIKSSI

In this section I discuss the design and implementation details of my extension to the WIKI concept. Because multimedia is a key component of my extension, I have coined the term WIKSSI, an acronym for *What I Know, See, and Say Is*.

6.1.1 Design

The WIKSSI system is an application that combines elements of multimedia and social crowdsourcing to facilitate the preservation of language through collaboration. In a similar

fashion to many other language learning systems, this system uses iconic representations of a concept to convey its meaning. The image is augmented with textual descriptions and categorical tags. Associated with each concept are one or more audio samples. Audio samples are the spoken form of the concept in a local language. In the case where no spoken audio samples exist, text-to-speech synthesis is used as a first approximation [69].

The WIKSSI system draws from recent innovation in many areas including collaboration, accessibility, usability, education and technology. At its core, the WIKSSI is a WIKI. Users are a primary source of information; they create new content and make existing content better. However, while a WIKI focuses primarily on text, WIKSSI directly incorporates, and relies on, richer forms of multimedia, including images and audio. By linking both textual and aural descriptions to a visual representation of some concept, users are able to make use of many resources during the learning process, according to learning styles.

Another crucial component of the system is that of the user. A WIKSSI system may start out empty or seeded with a minimal skeleton. It is primarily up to the users to fill out this skeleton with content. While a traditional WIKI stops with content creation and editing, the WIKSSI extends the roll of the user with additional privileges and responsibilities. Users have the responsibility of voting ("up-voting" and "down-voting") on, and tagging content. This allows the content to be dynamically ranked, categorized, and presented to other users in the most appropriate fashion. As users vote and tag content, they gain reputation. These two sources, namely user reputation and content voting, serve as authority sources for the system. For example it can be assumed that content with a high vote count or content that was submitted by a user with a high reputation is more credible than other content. This incentive-based, crowdsourcing model has become very popular in recent times. Many Web applications use a similar approach for generating content, the most prominent being Digg, StackOverflow, ServerFault and SuperUser. The WIKSII takes this one step further, combining collaboration with incentives and multimedia, all to create a rich and unique next-generation learning environment.

Table 6.1 shows several tools suitable for collaborative learning including CMS and WIKI systems. My proposed WIKSSI system attempts to improve on these systems by incorporating their specific strengths and adding new trends in modern computing such as multimedia, real-time content and crowdsourcing. Here, I specifically focus on context aware content, interoperability with assistive technology, and collaborative features of the WIKSSI compared to its counterparts.

	WIKI	**CMS**	**FirstVoices**	**VLL**	**WIKSSI**
Reciprocal Learning	often static scope, static feedback	static scope, some dynamic feedback	static scope, static feedback	static scope, some dynamic feedback	dynamic scope, dynamic feedback
Alternative Learning Styles	visual, verbal, aural, solitary	visual, verbal, aural, social, solitary	visual, verbal, aural, solitary	visual, verbal, aural, social, solitary	visual, verbal, aural, social, solitary
Alternative Knowledge Sources	text, images, audio, video, human	text, images, audio, video, human	text, images, audio, video, human	text, images, audio, video, human	text, images, audio, video, human, artificial intelligence
Alternative Information Format	text, images, hyperlinks	text, images, audio, video, human, hyperlinks	text, images, audio, video, animations	text, images, audio, RSS feeds, streaming video	text, images, audio, tags, emergent knowledge
Collaboration	community driven, community monitored	admin driven, admin monitored, coordination overhead	community driven, admin monitored, coordination overhead, collaborator credibility	admin driven, admin monitored, coordination overhead	community driven, community monitored, collaborator credibility

Table 6.1: Educational goal based analysis of existing infrastructures and WIKSSI

Context Aware Content

The WIKSSI system is a data-driven application, relying on its user base to add value by contributing content. An advantageous side effect of this is the trending topics that emerge through its use. Examples of this include trending topics on the Twitter platform [70], where current world events tend to dominate the information stream as they happen. This real-time component adds value to the WIKSSI learning environment.

Consider the 2010 Olympics held in Vancouver, Canada. While this area is bilingual in English and French, many foreign visitors require other language translations. Further, because the Olympics are a special event, many of the terms and concepts are unique. For example the phrases “photo finish”, “biathlon” or “carry the torch” require some context to be meaningful. Metadata associated with WIKSSI entries allow users to indicate this context when learning Olympic-specific concepts.

Figure 6.1: Demonstration of the WIKSSI language and tagging functionality

Figure 6.1 shows my current prototype of the WIKSSI. The centre column contains the main content for the application, housing the concepts and their associated samples. The outer columns provide tools for content discovery, two of which focus on trending

concepts. The Popular Tags widget, found in the rightmost column, presents a set of the most recent, highly used tags. The Popular Images widget, located in the leftmost column, displays a grid of recently high-viewed concepts.

Closely related to users finding trending concepts, are users contributing trending concepts. This dynamic behaviour is an advantage over static learning environments. With a static learning tool, providers have no way of anticipating what content will be required beyond basic concepts; they cannot react to trending topics without costly redeployment.

Another powerful learning feature is the differing contexts created by the relationship among tags. For example a user may want to learn about groups of concepts like foods or sports. The tag system supports filtering of this variety. The user begins by finding a concept within the WIKSSI system. Associated with each concept is a set of zero or more tags that have some relation to it. These tags have been added by other members of the community. By selecting a tag, the user is presented with other concepts in the WIKSSI categorized by that tag.

Compliance for Assistive Technology

For users with disabilities, consistent and predictable user interfaces are essential to ensure successful usability of a piece of software. The most effective way to meet this requirement is to design flexible software whereby the user interface is decoupled from backend logic, and to adhere to standards where ever possible. The WIKSSI attempts to achieve flexibility through a stateless, client-server architecture.

For example, many visually disabled and blind users rely on assistive technologies such as screen readers and screen magnifiers to interact with software. The WIKSII includes a Web-based implementation whereby users interact with the system through the browser. In past years, Web applications that used multimedia were often forced to rely on opaque technologies such as Flash. Assistive tools are often unable to interrogate Flash content, and thus cannot convey any useful information to its users. I instead chose to use the standards-compliant features provided by HTML 5 to deliver multimedia content. The WIKSSI system uses images to represent a concept, but alt tags are always provided so assistive tools can convey the images meaning in an alternate format. Also, audio is embedded with HTML 5 audio tags instead of Flash. This allows for the audio controls to be scripted and accessed programmatically (e.g. again with assistive tools).

Many users, particularly in developing regions, are beginning to consume digital content on low-cost, mobile devices instead of full-scale desktop computers. At least two

problems exist when porting desktop applications to mobile devices [28]. First, the amount of screen real estate on a mobile device is quite reduced compared to its desktop counterpart. Second, many new mobile device interfaces are touch-based, making them unusable by people with disabilities. I have implemented two native mobile applications, one on the iPhone platform and one on Android, that attempt to alleviate this problem. In contrast to the Web application that requires a high resolution to display its content, I have implemented a drill-down interface that presents content to the user in a hierarchical fashion. I have also included hooks within the application for onboard accessibility tools to interrogate the application for disabled users. For example, by providing alternate, text-based descriptions for widgets and pictures, iPhone OSs VoiceOver technology can assist visually disabled users as they navigate and use the application.

Collaboration

An important goal addressed by the WIKSSI is reciprocal learning. A conventional WIKI generates its corpus by users iteratively adding to, and editing material. In this case, the corpus itself becomes an authority on some topic, and the users who contributed to it are not credited individually. The WIKSSI works differently. A user first creates a concept, and then other users contribute by adding samples to that concept. Instead of aggregating these samples into a single corpus, the samples are voted on by the community. The samples with the most votes become the authority for that concept. Additionally, authority can come in differing amounts, depending on the class of the user. For example, a teacher or a community Elder submitting a sample would more credible than a student. To achieve reciprocal learning, the student first learns by listening to the existing samples. She then attempts her own version and contributes it back to the WIKSSI. That contribution is then voted on and ranked, providing feedback to the student.

6.2 Implementation

The WIKSSI implementation is highly flexible, extending beyond a typical Web application. The server exposes the WIKSSI data and functionality through a defined API, which can be called by a JavaScript-enabled Web browser or any internet capable application. The WIKSSI system is modelled after a typical client-server architecture. The server is implemented using Java servlets. The Spring [71] framework provides many additional features ranging from an Inversion of Control container and Internationalization, to database and

transaction support. The vast majority of application data is stored in a mySQL database and accessed through a Hibernate Object-Relational Mapping (ORM) layer. The one exception is audio files, which are both stored in the database as blobs and in the filesystem directly.

```
<!-- HTML 4.01 -->
<div id=container>
  <div id=header>
    the header
  </div>
  <divid=nav>
    some nav items.
  </div>
  <div id=sidebar>
    sidebar content
  </div>
  <div id=main>
    The main contents
  </div>
</div>

<!-- HTML 5 -->
<div id=container>
  <header>
    The header section
    <nav>
      the navigation
    </nav>
  </header>
  <aside>
    sidebar content
  </aside>
  <section>
    The main content
  </section>
</div>
```

Figure 6.2: Comparison of HTML 5 and HTML 4.01

Several client implementations of my prototype WIKSSI exist, including a browser-based version and two native mobile versions. The browser client is composed primarily of HTML, CSS and JavaScript, and incorporates several anticipated technologies surrounding HTML 5. As shown in Figure 6.2, new document layout tags are used instead of generic "div" elements. While not fully supported yet, these tags will greatly improve assistive softwares ability to find application content. Documents prior to HTML 5 were structured with generic "div" elements, which are distinguished only by their id and class attributes. Because application developers arbitrarily assign these attributes, it would be incorrect for assistive tools to make assumptions about their intent. HTML 5, on the other hand, employs tags that explicitly convey their meaning. For example, "nav" elements are used to hold navigational content, "aside" elements hold material indirectly related to the main

document, and “section” elements hold the primary content. Assistive tools can make use of these explicit structural components to help the user more quickly navigate to the desired location within the document.

```
<audio src=somefile.mp3
        controls
        alt=an audio recording />
```

Figure 6.3: HTML 5 Audio

Another critical component of the application is user-submitted audio. Through a consistent API and specification for required codecs, HTML 5 provides a standardized way of presenting audio to the user without the need for JavaScript or Flash, as shown in Figure 6.3. This is a great improvement over prior workarounds including Flash, third party plugins, MIME types and other non-standardized approaches. Instead of assistive software being baffled by foreign markup, they can at least convey to the user what the original content was through the alt attribute.

6.3 Cost Analysis

As in the case of BUSSPASS (Section 3) and collaborative learning software, the cloud has proven to be an appealing platform for application deployment. Instead of focusing on infrastructure and low-level system optimization, developers are able to place more resources in application development. A significant drawback of such an approach are the costs incurred when using cloud services. Financial resources are often limited for developing regions and organizations such as those that focus on accessibility, who are not servicing mass markets. Based on plan details from popular cloud service providers, application costs primarily consist of data storage and data bandwidth:

$$TotalCost = (DatabaseSize * StorageCost) + (DataTransferred * BandwidthCost)$$

where *DatabaseSize* and *DataTransferred* are measured in gigabytes, and *StorageCost* and *BandwidthCost* are measured in dollars per gigabyte. *TotalCost* is the monthly cost incurred from storing data and serving it to users. To provide a sense of how much a typical

application may cost, I performed an analysis based on the parameters of the WIKSII system. First, to calculate the amount of storage required, I considered the number of concepts, the number of samples per concept, and the average size of the samples (in gigabytes). This lead to the following equation for determining storage requirements:

$$DatabaseSize = concepts * samples * sampleSize$$

If an average sample size is assumed to be to be 50 kilobytes (0.00005 gigabytes), the system containing 50 concepts each having 10 samples, the total storage size would be 25 megabytes (0.025 gigabytes). Second, to calculate the amount of monthly bandwidth required, the size of the samples are considered, as well as the number of users and the number of requests they make per month:

$$DataTransferred = users * requests * sampleSize$$

Again, using an average sample size of 50 kilobytes, if 100 users each made 100 monthly requests, the total bandwidth used would be 500 megabytes.

With these parameters in place, assumptions can be made about typical usage scenarios to produce a lower and upper bounds for cost. For simplicitys sake the sample size should be held constant at 50 kilobytes. At the lower end a small, inactive user base and a minimal collection of concepts and samples are assumed. A user base of 100 users, making 200 requests per month, and collection of 100 concepts with 5 samples each, results in a data size of 25 megabytes and a monthly bandwidth usage of 1 gigabyte. At the upper limit a large user base of 10,000 active users, each making 1000 requests per month, and a large dataset consisting of 10,000 concepts with 100 samples each is assumed. This results in a data size of 50 gigabytes and a monthly bandwidth consumption of 500 gigabytes. Table 6.3 shows the monthly cost if these upper and lower bounded systems were to be deployed on various popular service providers.

	Monthly Storage Costs	Monthly Bandwidth Costs	Monthly Cost of Lower Bounded System	Monthly Cost of Upper Bounded System
Google App Engine	First GB free, then $0.15 / GB	First 30 GB free, then $0.12 / GB	0	$63.75
Amazon Simple DB	First GB free, then $0.25 / GB	First GB free, then $0.17 / GB	0	$97.08
Amazon RDS	$0.10 / GB (plus $0.10 / million IO requests)	$0.17 / GB	less than $1.00	$90.00
Amazon s3	$0.15 / GB	$0.17 / GB	less than $1.00	$92.50

Table 6.2: Data storage and bandwidth costs for popular cloud service providers

6.4 Data Usage and Timings

Arguably the most important part of the application is the user submitted audio. Accordingly, the quality of the upload process and accessing the audio is a major concern. Equally important is the cost of storing and transmitting this audio to users. Therefore, a balance between cost and quality is desired. Many of the low cost solutions such as Google App Engine and Amazon RDM (see Table 6.3) require the use of a database for data storage. However, studies have indicated that the use of a database to serve binary resources is an inferior approach to a simple filesystem solution [72] [73]. To test this claim, I performed the following experiments.

The WIKSSI system was preloaded with 40 English audio samples. The samples were encoded as mp3 and ranged in size from 35 kilobytes to 64 kilobytes. These samples were served in two fashions: directly from the servers filesystem as a linked resource, and from a mySQL database through a servlet. The database was configured in a standard / default configuration with no particular optimizations to the connection pool. Further, no optimized caching scheme was employed for either the filesystem or database. The Web server / servlet container used was Apache Tomcat. Both the servlet container and database server were deployed on the same system. To access the resources, the following two URL schemes were mapped:

- http://localhost:80805/audio/english/xxx.mp3
- http://localhost:8085/audioDB/xxx.mp3

In the first context, audio files were accessed directly on the server machine, at the relative path: audio/English/xxx.mp3 where xxx can be substituted for any existing audio file. The second context sees the components of the URL mapped to a Spring controller in the form of a servlet. The URL /audioDB is the mapping to this particular servlet, and xxx.mp3 is a variable used by the servlet handler. The servlet receives the incoming request, and uses the variable xxx.mp3 to look up an entry in the database. An existing database entry returns a byte array which represents the audio data for that name. The servlet then sets the appropriate content-type header and writes the byte array out as a stream. To the client both approaches are transparent. Http_load [74] is a simple command line tool that was used to test the two audio providing schemes. It is invoked as follows:

./http_load parallel CONCURRENT_USERS seconds TEST_TIME URL_TEXTFILE

The first option specifies how many simultaneous connections should be made to the server. I used this option to simulate the act of multiple users accessing the system at once. The second parameter specifies how long each test should last. The last argument is the name of a text file. This text file contains a list of the URLs the tool should attempt to fetch. I ran two series of tests. The first series of tests used a URL file containing URLs for accessing the 40 audio files directly from the filesystem. The second series contained URLs for fetching the files through the database. To reduce sensitivity to spontaneous background processes, the test instances for filesystem and database were interleaved.

The test contexts consisting of the following: three connection sizes were used. This simulated various numbers of concurrent users, namely 10, 50 and 100. My tests revealed that more than 100 threads produced vastly varying results from trial to trial. These sets of connection sizes were run 15 times each for duration of 5 seconds and then 10 seconds. Each set of connection sizes were also run 10 times with duration of 30 seconds. The tests were run on a Mac-based Intel computer with a Core 2 Duo processor running at 2.5 GHz, and 4 GB of RAM.

6.4.1 Timings

The results of my tests are summarized in Figures **??**. At low thread counts, which simulates relatively few concurrent users, the database implementation was able to fulfill more

requests than the filesystem. The database had an average throughput of about 28.4 requests per second, while the filesystem had a throughput of about 14.8 requests per second. As the number of concurrent users was increased, the database was negatively impacted, averaging a throughput of 6.64 requests per second. The filesystem became more productive, serving 29.4 requests per second. Overall, when going from 10 to 100 threads, the filesystem implementation saw a performance improvement of 129%. However, the database saw a reduction of 75.6%.

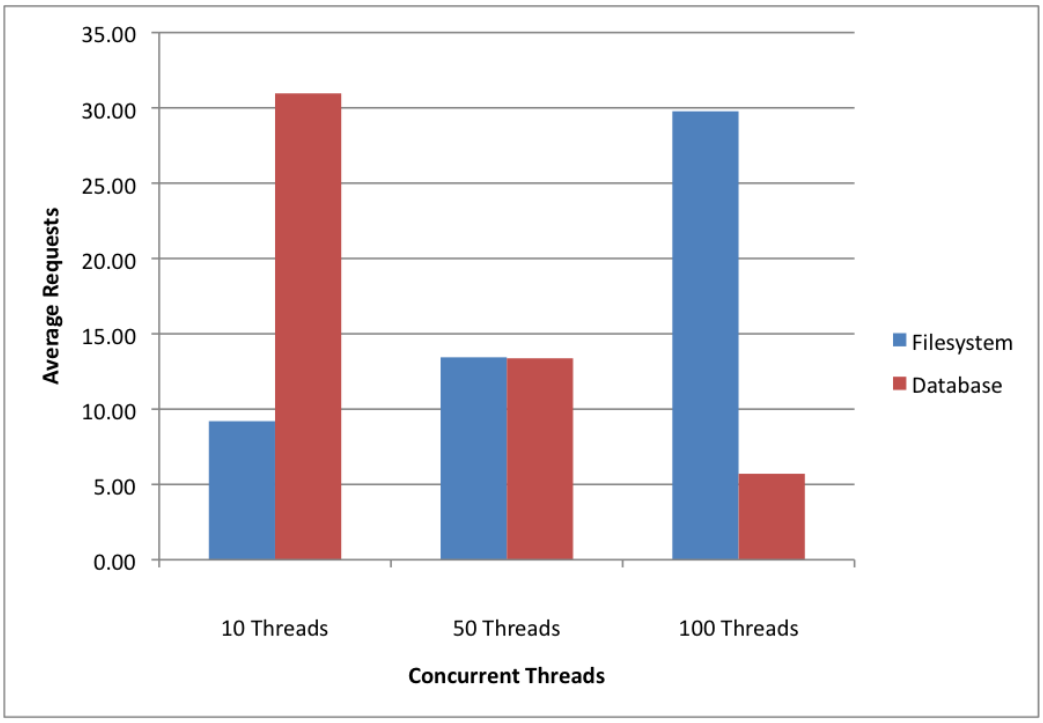

Figure 6.4: Number of Requests Filled for Filesystem and Database Resources Using a Test Duration of 5 Seconds

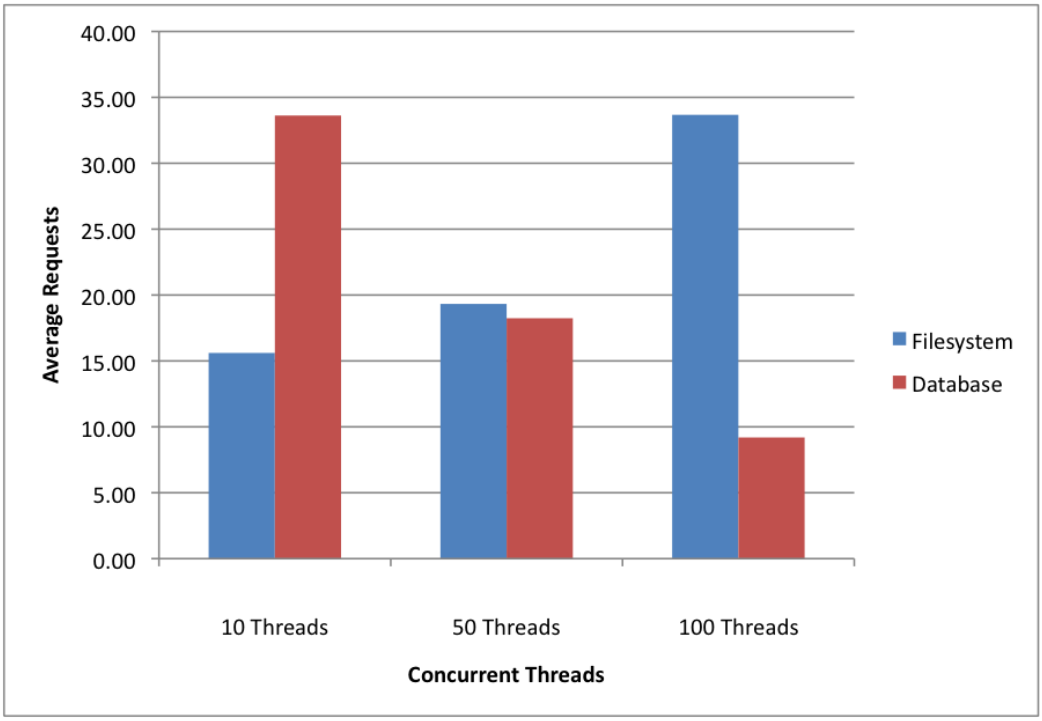

Figure 6.5: Number of Requests Filled for Filesystem and Database Resources Using a Test Duration of 10 Seconds

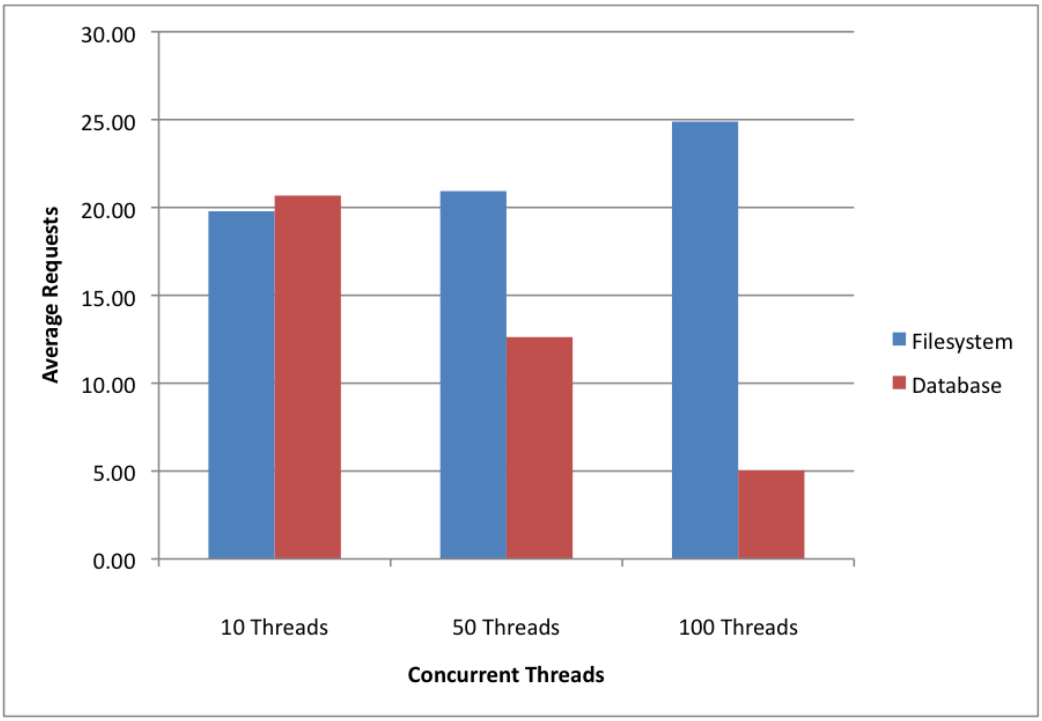

Figure 6.6: Number of Requests Filled for Filesystem and Database Resources Using a Test Duration of 30 Seconds

Chapter 7

Proposed Framework for Accessible Application Design

The case studies described in Chapters 3, 4, 5 and 6 describe applications whose intent is to enable users with disabilities and improve usability for able-bodied users through innovative user interfaces and modes of operation. While all achieve varying degrees of accessibility, striking similarities and differences exist in both the design and implementation of each. It is apparent that many factors contribute to these similar and dissimilar characteristics, from hardware limitations, to software development kits and support tools, to an overall understanding of user requirements. Producing accessible software as an afterthought can be a very costly undertaking, and developing truly innovative applications requires forethought well before coding begins. As such,Ihave compiled a set of guidelines, anti-patterns and ontologies, presented as an abstract framework, whose intent is to provide developers the support required in creating highly usable applications.

7.1 Motivation: Application Tiers and Platform Agnostics

When it comes to the interactions between users and software, distinct use cases have been established which separate mobile and desktop applications. Desktop applications are usually rich environments that provide the user with many features related to some domain (for example business, multimedia creation or entertainment). Mobile applications, on the other hand, tend to be much smaller and focused in terms of the functionality they provide. A typical iPhone user may install dozens of applications for specific tasks such as e-mail, driving directions or Web browsing. This reduced expectation of functionality is impacting

on developers because it allows them to choose alternate, often lighter-weight, runtimes and development environments for their applications.

My analysis of several mobile platforms has led us to define a three-tier ontology of applications which is illustrated in Figure 7.1. Built atop the operating system (OS) is a hierarchy of application environments: native applications, hosted applications and Web applications. The hierarchy is arranged with respect to the level of interaction a developer has to the OS and the underlying hardware features of the mobile device. At the base of the hierarchy are native applications. These programs are built with core libraries and executed within the native runtime of the OS. The next hierarchical tier are whatIrefer to as hosted applications. A hosted application is a piece of software that executes within a runtime not native to the underlying platform. Examples of such runtimes may be virtual machines (VM), for example the Java Virtual Machine [75] or Python [76] interpreter, or other runtimes like Flash or Silverlight. At the top of the hierarchy are Web applications. Web applications are typically implemented with Javascript, HTML and CSS, and are intended to be executed within a Web browser that supports these technologies.

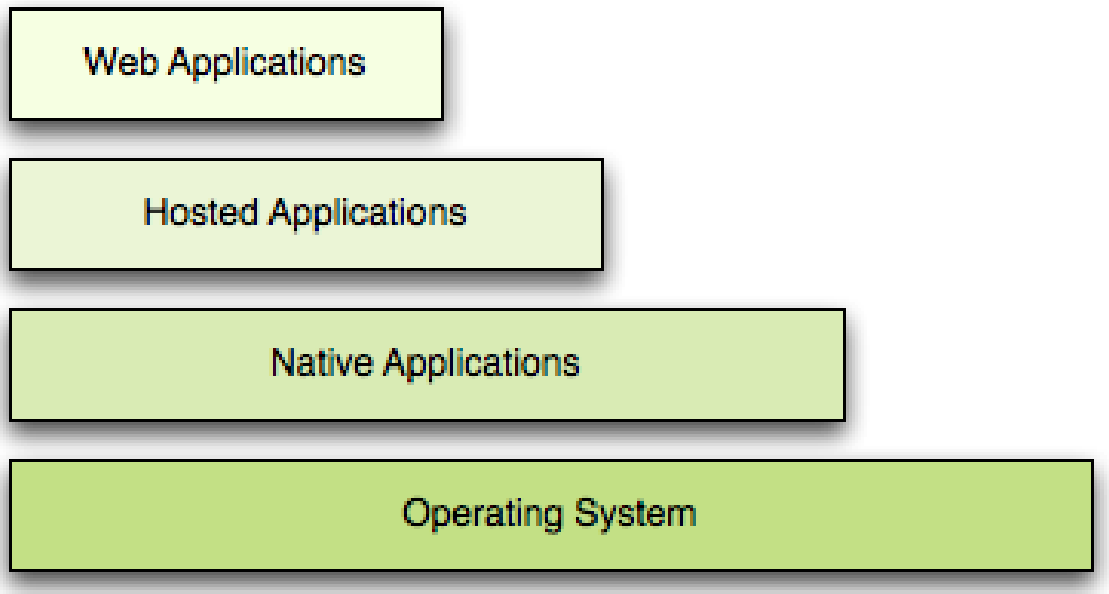

Figure 7.1: Hierarchy of mobile application environments

When choosing a tier for their application, developers must consider various tradeoffs. Web applications are generally easier to implement because they receive a high degree of support from the Web browser in which they execute. Also, because of Web standards,

these applications are relatively easy to port among the various Web browsers, allowing developers to reach a large audience. However, high portability and rapid application development (RAD) come at the expense of performance and control over the application. This is what native applications offer. Applications targeted at a specific platform have the most access to platform-specific features and can be optimized for better performance.

An excellent example of the development tiers can be found in an example by Nokia. Their *Sudokumaster: Designing a Flash Lite Game for Keypad and Touch Devices*[77] details the creation of a Sudoku game with Adobe's Flash Light, while their *S60 Platform: Sudoku Game Example* [78] details the steps involved in creating a Sudoku game that provides a consistent interface across the many screen resolutions and input methods found across different variants of the Symbian OS. Six screen resolutions are used by the application, from 128 x 160 pixels to 800 x 352 pixels, while maintaining displays that show the most important information for the current task. The game is played through different input methods, such as keypads and touch screens, depending on the device.

7.1.1 Extending Accessibility Guidelines Beyond Web Applications

When it comes to accessibility in this domain, Web apps are the most documented and rigorously tested type of application. This is likely due to the fact that Web applications have existed on the desktop for several years and much work has been done in building evaluation and tooling support to maximize their accessibility. The World Wide Web Consortium has put forth several documents dealing with everything from content creation guidelines, to recommendations for user-agents and evaluation tools.

Research into accessible Web applications has laid the ground work for the types of questions that need to be examined for these other tiers of mobile software. A more sophisticated W3C specification is the Accessible Rich Internet Application (WIA-ARIA) [7] guideline. This document is motivated by the observation that advanced Internet applications with dynamic interfaces and complex UI controls require unique guidelines, separate from traditional static HTML/CSS Web applications. Although intended for Rich Internet Applications, WIA-ARIA guidelines address many dimensions of mobile applications as well. Examples include drag-and-drop interactivity, updating Web-provided content in response to user actions, and complex UI widgets such as tree structures for navigation. However, as more hardware features are explicitly made available to lower-tier applications,Ican be more candid about the details that are included in the corresponding accessibility guidelines.

The US Department of Justice compiled a guideline aimed at evaluating the accessibility of software for users with disabilities [79]. This guideline is of interest because it examines software in a more general sense, as opposed to focusing specifically on the Web. Ihave taken this guideline and applied its relevance to each of the three tiers of software defined previously (Section 7.1). Table 7.1 is a reference to the 29 questions that make up the guideline. Each question is assigned one, two, or three check-marks, representing an independent developer's influence over that particular dimension of accessibility. For example, question one refers to whether or not the developer has control over all keyboard shortcuts in an application. This question was assigned two check-marks, excluding Web applications. Web apps are excluded in this case because the browser retains control of several keyboard shortcuts (i.e. the developer does not have complete control over this dimension).

Conclusions drawn from Table 7.1 shed light on the nature of the problem at hand: Though much work has been done to increase the accessibility of Web apps, they only tap into a fraction of the features and services available to a native or hosted application. Accordingly, current guidelines need to be extended to include these lower tiers, and increased support, whichIpropose in the form of a framework (Section 7.2) needs to be made available to developers so that they may create accessible software to its full potential.

Question	**A**	**B**	**C**
Does the software provide keyboard equivalents for all mouse actions, including buttons, scroll windows, text entry fields, and pop-up windows?	✓	✓	
Does the program provide clear and precise instructions for use of all keyboard functions as part of the user documentation?	✓		
Are instructions regarding keyboard use widely available for all users in your component?	✓		
Does the software have a logical tabbing order among fields, text boxes, and focal points?	✓	✓	✓
When navigating screens and dialog boxes using the keyboard, does the focus follow a logical tabbing order?	✓		

Is there a well-defined focal point that moves with keyboard navigation (e.g., can you use the arrow keys to navigate through a list followed by pressing the ENTER key or space bar to select the desired item)?	✓	✓	✓
Are shortcut keys provided for all pull-down menus?	✓		
Does the software support existing accessibility features built into the operating system (e.g., sticky keys, slow keys, repeat keys in Apple Macintosh OS or Microsoft Windows 95)?	✓		
If timed responses are present, does the software allow the user to modify the timing parameters of any required timed responses?	✓	✓	
Are all descriptions or labels for fields positioned immediately to the left or directly above the control, and do they end in a colon, so that it is easy for screen reading software to associate the labels with the corresponding fields?	✓	✓	✓
Does every window, object, and control have a clearly named label?	✓	✓	
Does the software application use standard controls rather than owner-drawn or custom controls?	✓		
Does the software have a user selectable option to display text on icons, i.e., text only icons or bubble help?	✓	✓	
Is the use of icons consistent throughout the application?	✓	✓	✓
Are menus with text equivalents provided for all icon functions or icon selections on menu, tool, and format bars?	✓	✓	✓
If there are audio alerts, are visual cues also provided?	✓	✓	✓
Does the software support the “show sounds” feature where it is built into the operating system?	✓		
Can the user disable or adjust sound volume?	✓		
If information is provided in an audio format, is it also capable of being displayed by the user in a visual format?	✓	✓	✓
Is the software application free of patterned backgrounds used behind text or important graphics?	✓	✓	✓

Can a user override default fonts for printing and text displays?	✓		
Can a user adjust or disable flashing, rotating, or moving displays?	✓	✓	✓
Does the software ensure that color-coding is never used as the only means of conveying information or indicating an action?	✓	✓	✓
Does the application support user-defined color settings system-wide?	✓		
Is highlighting also viewable with inverted colors?	✓	✓	✓
If the software application draws its own screen elements, does it pick up the size settings that the user has selected in the Control Panel?	✓		
Are all manuals and documentation provided in electronic format as well as ASCII text files, including text descriptions of any charts, graphs, pictures, or graphics of any nature?	✓		
Can a user choose to have any report generated by the software made available in a "print to ASCII file" format?	✓	✓	✓
Is special training provided for users with disabilities that will enable them to become familiar with the software and learn how to use it in conjunction with assistive technology provided as an accommodation?	✓		

Table 7.1: Software accessibility guidelines for users with disabilities - as put forth by the US Department of Justice, Civil Rights Division. A = native app support, B = hosted app support and C = Web app support

7.2 The Proposed Framework

Based on my analysis of current user interface architectures, platforms and hardware features, as well as the identification of the three-tiered nature of mobile software,Ibelieve a framework is necessary to assist developers in creating and evaluating accessible applica-

tions. The components ofmyproposed solution are illustrated in Figure 7.2.

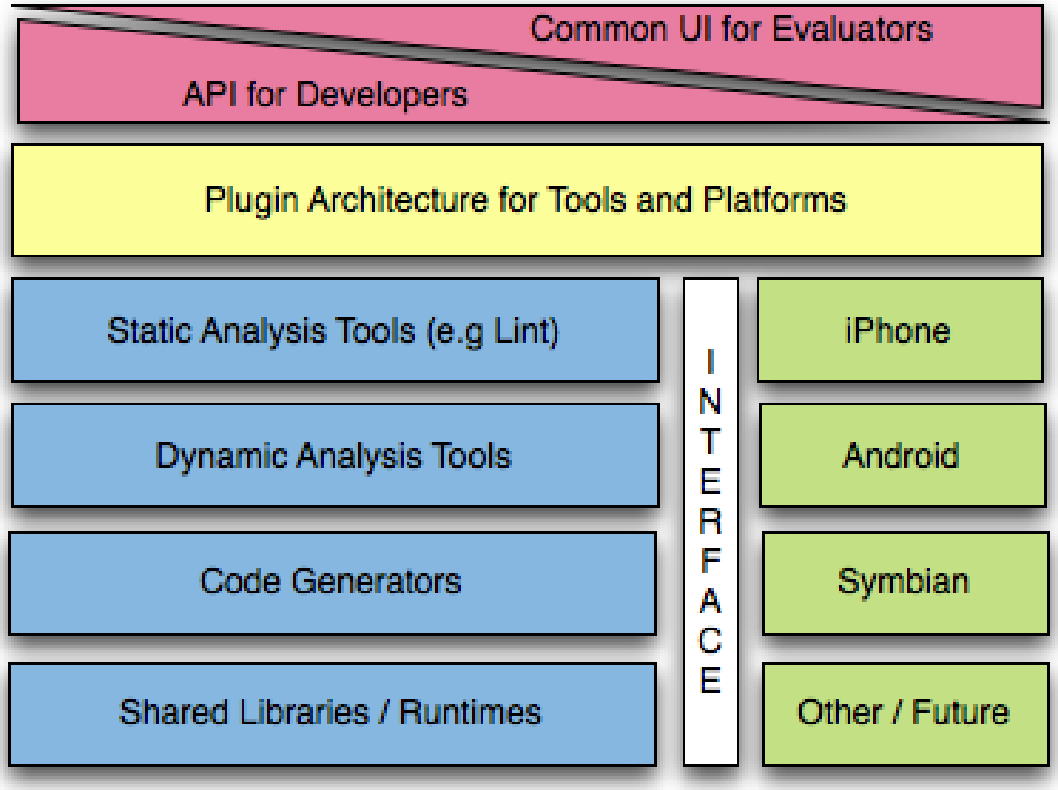

Figure 7.2: A block diagram of the proposed framework

The generic nature of the proposed framework allows for the inclusion of many software engineering methodologies currently being researched and refined, including code generation, static and dynamic analysis and runtime support for abstracting common accessibility features. The framework architecture ismyresponse to the belief that improving accessibility for mobile applications needs to happen throughout the various phases of the software development lifecycle. The cumulative contributions from requirements gathering, design, development, user testing and formal evaluation, together will produce software with superior accessibility features. Thus it ismyintent to provide a framework that facilitates each of these phases.

At the lowest level, the framework includes an internal interface to abstract the details of specific platforms where possible. This allows the framework to remain largely language agnostic, and expansible to future platforms. Making use of this common interface are the

framework components responsible for aiding the various software lifecycle phases mentioned previously. For example, static and dynamic analysis tools can be used to help an evaluator detect possible accessibility problems at both compile time and run time. Similarly, code generation tools can be used to aid developers by injecting Classes, methods or other software components that are capable of adding some dimension of accessibility to the application.

Chapter 8

Discussion and Evaluation

Thus far I have detailed four applications whose intent is to improve accessibility for some specific domain problem, and proposed a framework for facilitating developers in creating accessible software. This section focuses on evaluating and validating the feasibility and usefulness of such a framework. It also explores the ways in which these applications conform and deviate from the framework.

8.1 Case Study Analysis

In Chapter 1 I presented Tables 1.1 and 1.2 which summarized the accessibility characteristics and challenges associated with applications of each case study's domain. Here I present Table 8.1, which outlines the approaches taken to overcome the limitations of non-existent or partial third-party assistive support. In each example, several unique characteristics are highlighted that serve to improve overall usability and accessibility. BUSSPASS relies on the automatic reduction of unnecessary information to simplify content presentation. MarGrid Mobile takes advantage of the physical form factor of a mobile device to allow the user to navigate his music collection by touching different areas of the display. The mobile implementation of the implicit patching scheme achieves simplicity through and implied convention, as well as custom UI views that drastically reduce visual clutter and maximize display area. The WIKSSI circumvents limited mobile resources by utilizing cloud computing to store multimedia files, perform computationally-expensive text synthesis, and maintain crowd knowledge.

The approaches taken to address the lack of a dedicated screen magnifier are fairly consistent across domains: using large text, contrasting colour schemes, and presenting

less textual information in general.

For users who rely on text-to-speech synthesis, the lack of a operating system-supplied screen reader poses the greatest challenge to developers. One proposed solution is to link one's application against a dedicated TTS library. While this may appear to be a reliance on a third-party service, a destination is present which serves to support the argument of this thesis. A TTS library may be shared by several applications; however, by linking it directly, that library is tied to the application while in a running context. A consequence of this is the increased control by the developer about which content is read aloud, resulting in higher cohesion and semantic meaning among the presented content of the application.

	Transit Information Viewing	**Music Collection Browsing**	**Flowchart-Like Data Structures**	**Language Learning**
Use Case Implementation	BUSSPASS	MarGrid Mobile	Implicit patching framework	WIKSSI
Application specific accessibility enhancements	information reduction via automated lookup using GPS and clock	spacial / tactile navigation via two-dimensional surface, machine learning used to organize music	custom views and widgets reduce screen clutter	custom widgets provide specific functionality, reliance on cloud for processing reduces strain
Alternatives to screen magnifier	large native fonts, contrasting colour theme, less information presented	assisted browsing via machine learning, contrasting colour surface, reliance on touch instead of sight	drill-down on composites maximizes subsection of interest, no screen area sacrificed to show connections	majority of textual information presented as spoken word

Alternatives to screen reader	app-specific TTS lib, reduced information for reader to present, no possibility of malformed markup or style-only markup, semantic meaning maximized	app-specific TTS lib, information conveyed through touch rather than text that must be read, controlled through gestures (i.e. no need to find control widgets)	app-specific TTS lib, hierarchical / nested structure much easier to describe aurally than web-like structure	app-specific TTS lib, majority of content is spoken word audio
Alternatives to conventional desktop input	touchscreen, custom app-specific widgets, keyboard-mapped controls, controls via gestures, voice recognition, camera-based QR codes	touchscreen, physical dimensions of device provide orientation, control via gestures	touchscreen, custom app-specific widgets, keyboard-mapped controls, controls via gestures, voice recognition	touchscreen, custom app-specific widgets, keyboard-mapped controls, controls via gestures, voice recognition

Table 8.1: Case study features that compensate for absent accessibility support

8.2 Framework Analysis

This section describes the methodologies used during evaluation and analysis to support the proposed framework. First, I intend to show that existing projects can make use of such a framework, regardless of platform choice. To do so I chose three open source codebases, each targeting one of three platforms: iPhone, Android or Symbian. Because an analysis of a single codebase ported to the three platforms was not possible (I know of no such open source cross-platform application) I decided to choose three similar applications, all of which relate to some form of communication using a mobile device.

The iPhone and Android applications chosen, Tweetero [80] and Twitta [81] respectively, were Twitter clients. A calling app, FreeCaller [82], was the Symbian representative.

Cloc, a project analysis tool [83], was used to gather various metrics of each platform project. Table 8.2 summarizes the project attributes of interest.

	iPhone	**Android**	**Symbian**
Lines of Code	6096	3848	3182
Number of Files	38	25	27
MVC Employed	yes	yes	yes
Model Classes	19	13	16
View Classes	2	2	5
Controller Classes	17	7	5
Embedded Images	21	20	11
UI Config Files	12	15	15

Table 8.2: Metrics of the platform-specific projects

Regardless of platform choice, developers adhered closely to the well established Model-View-Controller (MVC) design pattern. This is comforting, in that developers accept the importance of separation among application content, functionality and presentation. I also found that several platforms store their UI descriptions in external resource files. In many cases the format is a derivative of XML. Many open source tools exist to evaluate the accessibility of Web documents (which are in XHTML format). The following section shows how current Web evaluation tools can be modified to detect and/or modify these XML-based application resource files.

8.2.1 Resource File Generation and Modification

To illustrate how the proposed framework could be leveraged to modify existing codebases, I present the modification of a simple Android application. I began with NotepadCodeLab, a tutorial program provided by Google to demonstrate the construction of a simple application [84]. NotepadCodeLab is a list-based app that allows the user to insert and view notes. A screenshot of the unmodified application running on an emulator is shown in Figure 8.2.1. The original version of the application displays list items in regular-size font, making them difficult to read for people with vision disabilities.

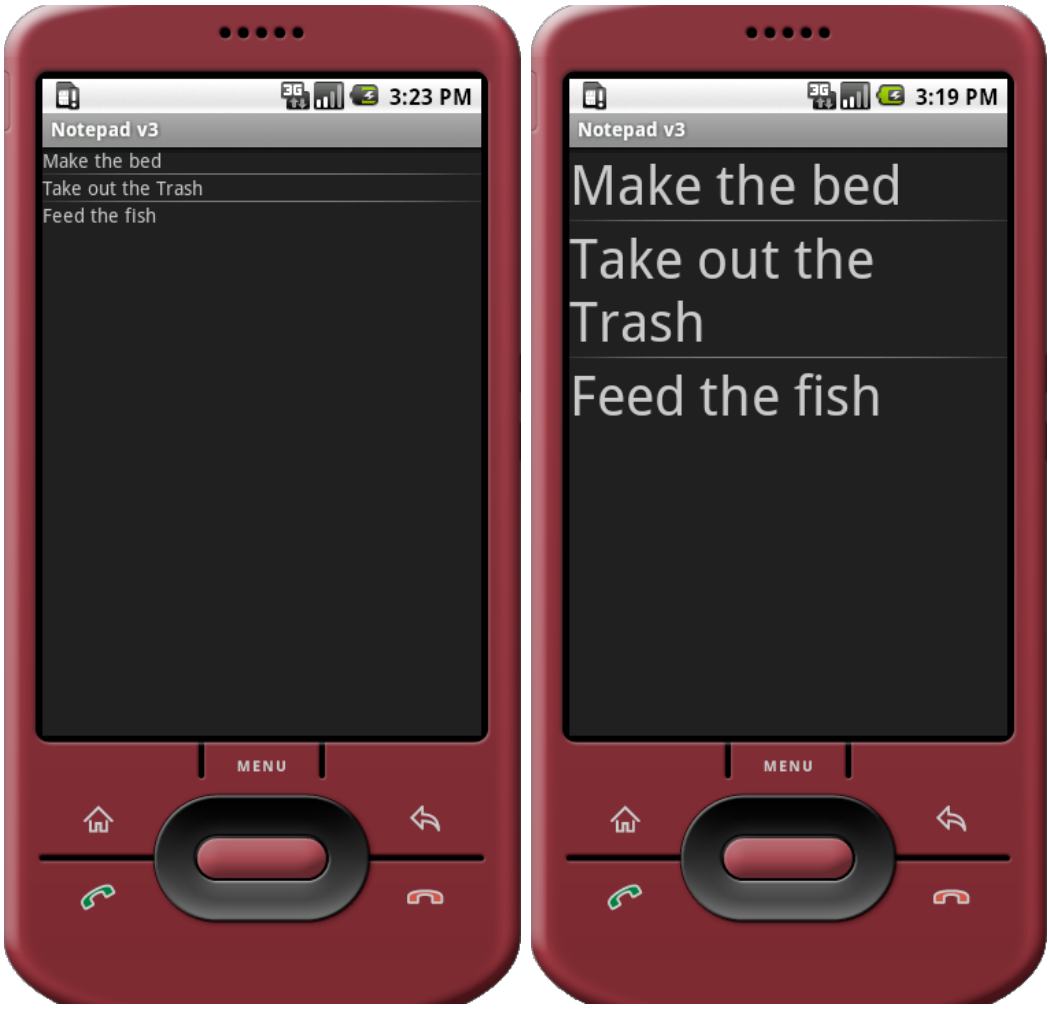

Figure 8.1: A comparison of the Android application NotepadCodeLab

As was described in Section 2.3.2, the logic of an Android application is implemented in Java sourcecode files, while the UI is often specified in an XML resource file:

`TextView` is a View object used by the application to show the text of the list items.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android_URL
    android:id="@+id/text1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="40px"/>
```

Figure 8.2: A user interface resource file for the NotepadCodeLab Android application

To override the default text size of the list items, I added the extra attribute: `android:textSize="40px"`. The result of this modification is shown in Figure 8.2.1.

The XML structure of Android resource files (as well as iPhone and other platforms) allows for tools with XML parsing capabilities to easily make these types of modifications.

8.2.2 Embedded Images

A common source of inaccessibility in software, with respect to the visually disabled and the blind, is the semantic meaning associated with embedded images. Images of culturally accepted icons (for example a right triangle to indicate a play button) are often inserted in place of a textual description. The functionality of an application is severely reduced if the user cannot learn about the user interface.

I examined each of the embedded images of the evaluation projects and assigned them the label 'aesthetic', indicating that the image has little to no intrinsic meaning, or 'semantic' if the image conveys essential information about the user interface to the user. These observations are listed in Table 8.3.

	iPhone	**Android**	**Symbian**
Aesthetic Images	7	14	8
Semantic Images	14	6	3

Table 8.3: Number of embedded images described as having semantic meaning or simply visual appeal

Although all of the applications make use of images to portray functionality to some degree, it would be erroneous to make absolute conclusions based on the numbers. Several mechanisms exist to complement an image's information in alternate formats. These include labels, tooltips, alt tags, etc. The determination as to whether or not these mechanisms have been implemented would have to be made via static analysis of source code or runtime observation.

Chapter 9

Conclusion and Future Work

This thesis provided background on the current state of accessible software as well as a survey that outlined contemporary capabilities of mobile hardware and software. Four case studies were presented, each of which aimed to remove usability hinderances from domain specific problem. First BUSSPASS was discussed. BUSSPASS is a Web and mobile tool for facilitating the lookup of transit scheduling information. Next MarGrid Mobile, a touch-based, self-organized music navigation system was presented. Third, a new model for constructing information networks based on implicit patching was described. Finally WIKSII, a highly-dynamic, user-propelled, multimedia learning application was put forth. Based on these case studies, as well as research we had compiled on accessibility and mobile applications, a framework was proposed whose intent was to aid developers in creating highly-usable software. Finally, we provided support for the feasibility and usefulness of such a framework, as well as document the adherence and separation of the case study applications to the framework.

9.1 Contribution

The contribution of this thesis is broad in scope. Each case study project presented is an avenue of study in its own right, and this thesis has served, partially, to document these projects as a starting point for further study. More importantly, however are the overarching trends and themes extracted from the case studies, and their application to the proposed framework. We have stated that the creation of accessible software, especially mobile software, can be a challenging task. In some cases, weaving accessibility features into existing applications, relying on underlying operating system services, or delegating to

third party assistive technology appears to be the most economical and favoured approach. Still, niche applications whose sole purpose is to facilitate accessibility, through esoteric user interfaces and unique modes of operation, may be the best approach to solve certain domain specific problems.

9.2 Future Work

Several possibilities exist for future work related to the case studies and framework put fourth in this thesis. BUSSPASS is currently implemented as a Web application that serves transit data to several types of clients including iPhone, Android or Web browser. The most challenging aspect of this project is gathering data to present to the clients. As mentioned in Chapter 3, screen scraping is a less than optimal solution as it requires significant development time to construct the required data gathering algorithms. A most desirable solution would be affiliate the project with existing data gathering services such as Google Transit.

The potential for MarGrid Mobile lies with the ability to port it to other platforms. It is yet to be determined whether or not the iPhone is a suitable mobile device for users with disabilities. Newer, larger devices such as the Apple iPad, or Android-based devices should be considered as possible alternatives.

The implicit patching model for information networks is an open-ended solution for many types of domain-specific applications. A generic framework that is able to represent different types of information networks is the primary objective. A concrete extension of the framework would see Marsyas networks built graphically with such a tool.

The next logical step in the WIKSSI project is to test its scalability. This service was intended to provide education for large, and possibly remote, user bases. As such, a cloud-based implementation would be appropriate.

As for our proposed framework, the move from abstract to concrete is the most prudent next phase. Platform-specific evaluation and code generation tools should be developed to complement existing guidelines and ontologies.

Bibliography

[1] (Retrieved March 3 2010) What makes a good user interface. [Online]. Available: http://uw714doc.sco.com/en/SDK_vtcl/vtclgN.style_goodui.html

[2] K. A. Butler, R. J. K. Jacob, and B. E. John, "Human-computer interaction: introduction and overview," in *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 1999, pp. 100–101.

[3] Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong, and Verplank, "Acm sigchi curricula for human-computer interaction," Web, ACM SIGCHI 1996.

[4] J. Neilsen, *Usability Engineering*. Morgan Kaufmann Publishers, 1994, no. ISBN 0-12-518406-9.

[5] B. Shneiderman, *Software psychology: Human factors in computer and information systems (Winthrop computer systems series)*. Winthrop Publishers, 1980.

[6] D. Z. Fleischer, *The disability rights movement: From charity to confrontation*. Temple University Press, 2001.

[7] (Retrieved March 3 2010) World wide web consortium. [Online]. Available: http://www.w3.org/

[8] (Retrieved March 3 2010) W3c wai guidelines. [Online]. Available: http://www.w3.org/WAI/

[9] (Retrieved March 3 2010) W3c wcag guidelines. [Online]. Available: http://www.w3.org/TR/WCAG10/

[10] (Retrieved March 3 2010) W3c atag guidelines. [Online]. Available: http://www.w3.org/TR/WAI-AUTOOLS/

[11] (Retrieved March 3 2010) W3c uaag guidelines. [Online]. Available: http://www.w3.org/TR/WAI-USERAGENT/

[12] (Retrieved March 3 2010) W3c xag guidelines. [Online]. Available: http://www.w3.org/TR/xag

[13] (Retrieved March 3 2010) W3c aria guidelines. [Online]. Available: http://www.w3.org/WAI/intro/aria

[14] (Retrieved March 4 2010) Voice extensible markup language (voicexml). [Online]. Available: http://www.w3.org/TR/voicexml20/

[15] G. Reitmayr and D. Schmalstieg, "Location based applications for mobile augmented reality," in *AUIC '03: Proceedings of the Fourth Australasian user interface conference on User interfaces 2003*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2003, pp. 65–73.

[16] (Retrieved March 3 2010) Umts. [Online]. Available: http://www.umts-forum.org/

[17] (Retrieved March 3 2010) Gsm. [Online]. Available: http://gsmworld.com/

[18] (Retrieved March 3 2010) Cdma. [Online]. Available: http://www.cdg.org/

[19] Apple. Interface builder user guide. [Online]. Available: http://developer.apple.com/documentation/developertools/Conceptual/IB_UserGuide/Introduction/Introduction.html

[20] Google. What is android? [Online]. Available: http://developer.android.com/guide/basics/what-is-android.html

[21] Google android blog. adobe announces flash for android at mobile world congress in barcelona. [Online]. Available: http://www.googleandblog.com/adobe-announces-flash-for-android-at-mobile-world-congress-in-barcelona/3785/

[22] K. Rush, "Getting started with adobe flash lite," 2007.

[23] "Silverlight for mobile : The official microsoft silverlight site."

[24] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959. [Online]. Available: http://www.research.ibm.com/journal/rd/033/ibmrd0303B.pdf

[25] (Retrieved March 3 2010) Google transit. [Online]. Available: http://www.google.com/intl/en/landing/transit/#mdy

[26] (Retrieved March 3 2010) Translink. [Online]. Available: http://www.translink.ca/

[27] D. Minfie and Y. Coady, "Busspass: Composite-services to improve quality of life for persons with visual disabilities." Bejing, China: MobEA, 2008.

[28] D. Minifie and Y. Coady, "Getting mobile with mobile devices: using the web to improve transit accessibility," in *W4A '09: Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A)*. New York, NY, USA: ACM, 2009, pp. 123–126.

[29] (Retrieved March 4 2010) Qr code standardization. [Online]. Available: http://www.denso-wave.com/qrcode/qrstandard-e.html

[30] (Retrieved March 4 2010) Google 411. [Online]. Available: http://www.google.com/goog411

[31] (Retrieved March 3 2010) Pollux. [Online]. Available: http://polluxapp.com/

[32] (Retrieved March 3 2010) Amplifind. [Online]. Available: http://www.amplifindmusicservices.com/

[33] (Retrieved March 3 2010) Last fm. [Online]. Available: http://www.last.fm/

[34] (Retrieved March 3 2010) Music brainz. [Online]. Available: http://musicbrainz.org/

[35] F. Vignoli, "Digital music interaction concepts: A user study," in *ISMIR*, 2004.

[36] F. Bentley, C. Metcalf, and G. Harboe, "Personal vs. commercial content: the similarities between consumer use of photos and music," in *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*. New York, NY, USA: ACM, 2006, pp. 667–676. [Online]. Available: http://dx.doi.org/10.1145/1124772.1124871

[37] S. J. Cunningham, N. Reeves, and M. Britland, "An ethnographic study of music information seeking: implications for the design of a music digital library," in *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 5–16. [Online]. Available: http://portal.acm.org/citation.cfm?id=827142

[38] E. Pampalk and W. Dezember, "Islands of music - analysis, organization, and visualization of music archives," 2001.

[39] R. Gulik and F. Vignoli, "Visual playlist generation on the artist map," in *in Proc. of the ISMIR Intl. Conf. on Music Information Retrieval*, 2005, pp. 520–523.

[40] M. Goto and T. Goto, "Musicream: New music playback interface for streaming, sticking, sorting, and recalling musical pieces," in *In Proceedings of the 6th International Conference on Music Information Retrieval*, 2005, pp. 404–411.

[41] M. Torrens and J. lluís Arcos, "Visualizing and exploring personal music libraries," in *In Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 04*, 2004, pp. 421–424.

[42] G. Tzanetakis, M. S. Benning, S. R. Ness, D. Minifie, and N. Livingston, "Assistive music browsing using self-organizing maps," in *PETRA '09: Proceedings of the 2nd International Conference on PErvsive Technologies Related to Assistive Environments*. New York, NY, USA: ACM, 2009, pp. 1–7.

[43] G. Tzanetakis and P. Cook, "Marsyas: a framework for audio analysis," *Org. Sound*, vol. 4, no. 3, pp. 169–175, December 1999. [Online]. Available: http://dx.doi.org/10.1017/S1355771800003071

[44] (Retrieved April 3, 2010) Marsyas and iphone os. [Online]. Available: http://www.noisyair.com/blog/2009/7/29/marsyas-and-iphone-os.html

[45] (Retrieved March 29 2010) Wikipedia. [Online]. Available: http://wikipedia.org

[46] (Retrieved March 29 2010) Moodle. [Online]. Available: http://moodle.org

[47] (Retrieved March 29 2010) Blackboard. [Online]. Available: http://blackboard.com

[48] (Retrieved March 29 2010) Connex. [Online]. Available: https://connex.csc.uvic.ca/portal

[49] (Retrieved April 3 2010) Firstvoices. [Online]. Available: http://www.firstvoices.com/

[50] (Retrieved April 3 2010) Virtual learning lodge. [Online]. Available: http://northislandvll.ca/

[51] J. Ball and S. McCaffrey, "Connectivity scan and survey report," School of Child and Youth Care, University of Victoria, MAE ASPF, 2006-2007.

[52] (Retrieved March 29 2010) Facebook. [Online]. Available: http://facebook.com

[53] (Retrieved March 29 2010) Facebook. [Online]. Available: http://secondlife.com/

[54] B. Alexander, "Web 2.0: A new wave of innovation for teaching and learning?" *Educause Review*, vol. 41, no. 2, pp. 32–44, 2006. [Online]. Available: http://www.educause.edu/apps/er/erm06/erm0621.asp

[55] P. Anderson, "What is web 2.0? ideas, technologies and implications for education," Tech. Rep., 2008. [Online]. Available: http://www.jisc.ac.uk/media/documents/techwatch/tsw0701b.pdf

[56] D. Brabham. (Retrieved March 29 2010) Crowdsourcing as a model for problem solving. [Online]. Available: http://con.sagepub.com/cgi/content/abstract/14/1/75

[57] A. Omar, D. Rose, and B. Stewart. (Retrieved March 29 2010) Crowdsourcing for relevance evaluation. [Online]. Available: http://portal.acm.org/citation.cfm?id=1480506.1480508

[58] J. Bradley, R. Lancashire, and A. Lang. (Retrieved March 29 2010) The spectral game: Leveraging open data and crowdsourcing for education. [Online]. Available: http://www.jcheminf.com/content/1/1/9

[59] (Retrieved March 29 2010) Stack overflow. [Online]. Available: http://stackoverflow.com

[60] (Retrieved March 29 2010) Server fault. [Online]. Available: http://serverfault.com

[61] (Retrieved March 29 2010) Digg. [Online]. Available: http://digg.com

[62] (Retrieved March 29 2010) How crowdsourcing is helping in haiti. [Online]. Available: http://www.newscientist.com/article/mg20527453.600-how-crowdsourcing-is-helping-in-haiti.html

[63] (Retrieved March 29 2010) Haitians in crisis. [Online]. Available: http://www.cmu.edu/homepage/society/2010/winter/haitians-in-crisis.shtml

[64] J. Tang, S. Yan, R. Hong, G.-J. Qi, and T.-S. Chua, "Inferring semantic concepts from community-contributed images and noisy tags," in *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*. New York, NY, USA: ACM, 2009, pp. 223–232.

[65] F. Luqman and M. Griss, "Overseer: A mobile context-aware collaboration and task management system for disaster response." C5, 2010.

[66] (Retrieved March 29 2010) Html 5. [Online]. Available: http://www.w3.org/TR/html5/

[67] (Retrieved March 29 2010) Apple voiceover. [Online]. Available: http://www.apple.com/accessibility/iphone/vision.htm

[68] (Retrieved March 29 2010) Google android tts. [Online]. Available: http://code.google.com/p/eyes-free/

[69] (Retrieved March 29 2010) Google translate tts. [Online]. Available: http://translate.google.com/

[70] (Retrieved March 29 2010) Twitter. [Online]. Available: http://twitter.com

[71] (Retrieved March 29 2010) Spring framework. [Online]. Available: http://springsource.org

[72] (Retrieved March 23 2010) To blob or not to blob: Large object storage in a database or a filesystem? [Online]. Available: http://arxiv.org/abs/cs.DB/0701168

[73] F. Tian, D. J. DeWitt, J. Chen, and C. Zhang, "The design and performance evaluation of alternative xml storage strategies," *SIGMOD Rec.*, vol. 31, no. 1, pp. 5–10, 2002.

[74] (Retrieved March 23 2010) Http_load. [Online]. Available: http://acme.com/software/http_load/

[75] (Retrieved March 15 2010) Java. [Online]. Available: http://java.sun.com

[76] (Retrieved March 15 2010) Python. [Online]. Available: http://python.org

[77] "Sudokumaster: Designing a flash lite game for keypad and touch devices," 2008.

[78] (Retrieved March 15 2010) Symbian s60 soduko game example. [Online]. Available: http://ncsp.forum.nokia.com/download/?asset_id=http://sw.nokia.com/id/d6f6f8d4-06cf-453c-9347-fb12c2541288/S60_Platform_Sudoku_Game_Example_v1_0_en.zip

[79] U.s. department of justice. section 508 - software. [Online]. Available: http://www.usdoj.gov/crt/508/report/software.htm

[80] (Retrieved March 15 2010) Tweetero. [Online]. Available: http://code.google.com/p/tweetero/

[81] (Retrieved March 15 2010) Twitta. [Online]. Available: http://code.google.com/p/twitta/

[82] (Retrieved March 15 2010) Twitta. [Online]. Available: http://sourceforge.net/projects/freecaller/

[83] (Retrieved March 15 2010) Cloc. [Online]. Available: http://cloc.sourceforge.net/

[84] Google. (Retrieved March 15, 2010) Android notepad tutorial. [Online]. Available: http://developer.android.com/guide/tutorials/notepad/index.html