Portable Digital Libraries on an iPod

David Bainbridge,† Steve Jones,† Sam McIntosh,† Matt Jones‡ and Ian H. Witten†

†Department of Computer Science
University of Waikato
Hamilton, New Zealand
{davidb, stevej, sjm64, ihw}@cs.waikato.ac.nz

‡Department of Computer Science
University of Swansea
Swansea, U.K.
matt.jones@swansea.ac.uk

ABSTRACT
This paper describes the facilities we built to run a self-contained digital library on an iPod. The digital library software used was the open source package Greenstone, and the paper highlights the technical problems that were encountered and solved. It attempts to convey a feeling for the kind of issues that must be faced when adapting standard DL software for non-standard, leading-edge devices.

Categories and Subject Descriptors
H.3.7 Digital Libraries, H.5.2 User Interfaces.

General Terms
Design, Experimentation.

Keywords
Mobile Digital Libraries, Open Source, iPod, Greenstone.

1. INTRODUCTION
Imagine being able to carry a library around in your pocket. Full fingertip access through searching and browsing to millions of items: text, image, audio, video, wherever you are. Really, *wherever* you are: no need to be within a wireless hotspot, or mess around with ISP registration (and worry about how much is it costing). No waiting for rich-media content to be transferred to your device, not to mention the accompanying rapid depletion of your battery power that goes with all that communication. Everything travels with you: no flight restrictions on planes; no worrying about connectivity in other countries. Your own personal copy of a large digital library, right there in your pocket.

Portable devices that are optimized for music and video, such as the iconic iPod, have vast amounts of local storage. However, they are also highly tailored in the software functionality they provide. Consequently we have considered whether it is possible to subvert personal multimedia devices to store entire digital library collections and take on the digital library functionality more akin to what is possible with a general-purpose PDA in a client-server configuration.

In this paper we describe how we developed an experimental prototype that explores this idea. First we provide details of the operating system, iPod-Linux, upon which the work rests. We also give a brief review of Greenstone, the digital library software that we fused with this minimalist operating system, before discussing developments necessary for it to run in the iPod environment.

JCDL'08, June 16–20, 2008, Pittsburgh, Pennsylvania, USA.

2. SOFTWARE FOUNDATION
iPod-Linux[1] is an open source project, launched in 2004, with the aim of porting Linux to the various generations of iPod. Based on a default installation—which is achieved through a simple "Wizard" style graphical interface—a user experiences iPod-Linux by interacting with a top-level application called Podzilla, which serves as a substitute for the Apple iPod interactive menu application. Traversing a hierarchy of menus, many of the same features are available. For example, users can launch playlists, artist lists, etc. in the iPod's iTunes database and have them play in the background while accessing further functions.

Podzilla is designed around a modular architecture that is highly configurable. Example modules include games such as Mastermind and Space Invaders, and utilities such as a calculator and color picker. Even the first-person shoot-'em-up game Doom has been ported!

iPod-Linux is based upon uClinux, a minimalist Linux distribution tailored for micro-controllers without memory management units. Successful ports to first-, second- and third-generation iPods have already been achieved. For more recent models (fourth, fifth, photo, video, nano) the core operating system functionality is provided but kernel modules for interacting with devices are less well-developed. Two omissions of note are audio input (i.e., recording) and network communication (Firewire/USB). As a result, these models are not (currently) officially supported by the iPod-Linux project. However, in practical terms we have found that the well-documented installation procedures can be followed for these models with the same results as for the supported models. At the time of writing, the second and third generation iPod-nanos and touch iPods do not run Linux at all.

Apple do not disclose full technical details of the iPod, and as one might expect hitches can occur when messing around with

[1] *www.ipodlinux.org*

components at this level—although these are rare in our experience. Apple provides software (initially a separate program, but now a feature of iTunes) capable of returning an iPod to its factory settings, so if the system becomes too badly scrambled it is easy to reformat it as a pure Apple-only device, upon which Linux can be re-installed.

Developing software for iPod-Linux requires a cross-compiler that is run on a desktop machine (the host). The generated executables are then copied across to the iPod (the target). When development of an iPod application (or Podzilla module) is complete, the executables would ultimately be bundled together and included as part of the installer.

Greenstone

Greenstone is an open source digital library toolkit [1]. Used out of the box it provides the ability to create collections of digital content, to display the content in a web browser and to access and search the collections that have been built. Each collection can be individually designed both in terms of appearance and search and browse functionality.

Countless digital libraries have been built with Greenstone since its public release on SourceForge in 2000: from historic newspapers to books on humanitarian aid; from eclectic multimedia content on pop-artists to curated first editions of works by Chopin; from scientific institutional repositories to personal collections of photos and other document formats. All manner of topics are covered: the black abolitionist movement, bridge construction, flora and fauna, the history of the Indian working class, medical artwork, shipping statistics—these are just a random selection.

3. CREATING AN IPOD MODULE

To develop a Greenstone iPod module some alterations to the runtime system were required. Greenstone operates via a CGI program intended to be run by a web server in response to each user request/query; it produces web pages in raw HTML that are rendered in a web-browser. Basically the runtime system needed to be re-cast as a program intended to operate continuously, rather than being re-invoked by the web server for each individual operation. In effect, the standard client-server mode of operation, with a web browser interface communicating with a web server, had to be overridden in favor of a continuously-running program.

A taste of the development process

Greenstone is written in the C++ language, and the first task we gave ourselves was seemingly simple: get the CGI programming running from the command line of the iPod. We take the trouble to describe the problems encountered here because although the details may be relatively uninteresting except to very technical people, the moral is not: arcane problems arise when retargeting consumer hardware to perform digital library functions, problems which require extensive research and painstaking experimentation to solve.

In fact, the software compiled easily. The cross-compiler for the iPod is based around *gcc*/*g++*, which happens to be what we have been using throughout the development of Greenstone. We were, however, disappointed by the result of running the compiled program: a segmentation fault occurred whenever printing or file input/output was attempted. Some of these issues were documented on the iPod-Linux web site; others we had to resolve from first principles.

First the well documented issues. There are in fact two cross-compilers in use for iPod-Linux. One, a fairly old incarnation of the compiler by today's standards, is required to compile the version of the Linux kernel used; the other is used mostly for iPod applications. The old version does not implement C++ file input-output (as opposed to C file input/output) correctly, which accounted for a segmentation fault occurring any time we tried to access a file. Moving to the newer version of the compiler solved this problem, but introduced the new complication that any printing to the screen (using the C++ objects *cout* and *cerr*) immediately produced a segmentation fault. Perversely, C++ style file input/output was working perfectly, as was printing to the screen using the C (*printf*) command. A series of small test programs were devised to shed light on the problem, and these led to the following conclusion. The key issue was that global objects were not being initialized, and while for user-defined global objects one can explicitly compensate for this, the same is not true for system defined global objects such as *cout* and *cerr*.

Few iPod-Linux developers use C++, and presumably the problem had not come to light before because those that do are using C-style print commands in their C++ code. The C++ standard explicitly allows the two approaches to be mixed freely, and while the C method is more arcane, it is an attractive choice because, once mastered, it is a more concise and faster way to achieve results. The Greenstone codebase consistently exploits the C++ object oriented paradigm throughout, leading to its downfall in this case.

In order to solve this problem it seemed neither practical nor desirable to replace all *cout*/*cerr* lines with their C equivalents. After some investigation we decided to use part of the C++ standard that governs the mixing of the two styles. By default, the C and C++ approaches are defined to be synchronized: a *cout* followed by a *printf* followed by a second *cout* strictly preserves the order of printing. The standard allows for strict synchronization to be relaxed. Placing an appropriate command at the beginning of the main program that instructed the two forms to become desynchronized had the important side-effect of *reinitializing* the *cout* and *cin*. With this modification included, the command-line test of Greenstone ran without fault.

Recasting Greenstone as a Podzilla module

The next step was to recast Greenstone as a module for Podzilla, allowing it to run continuously. While conceptually a more challenging problem, it in fact took us less time than resolving the *cout/cerr* problem. The resulting program was then added to the iPod's hierarchical menu system of modules that the user traverses in order to locate and execute application programs. Figures 1 and 2, which we discuss in more detail below, give the flavor of interacting with the digital library..

The iPod module works directly from the files that are generated by building a collection on a host machine running the standard Greenstone software. Through a configuration file in the Podzilla modules folder, the module is instructed whereabouts in the file system the Greenstone home file area resides. Once that is done, the rest of the process in automatic. This means that standard Greenstone collections can be transferred to the iPod, placed in

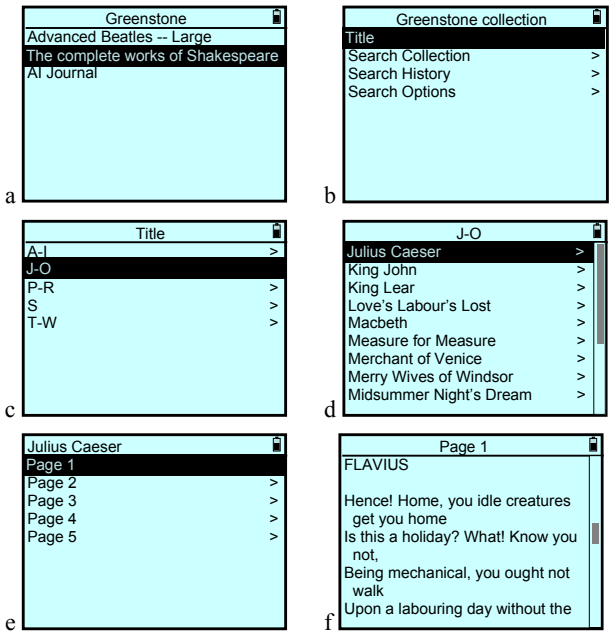

Figure 1 Browsing by title around the Shakespeare plays

the appropriate location, and displayed there without any intervention or conversion.

When it is run, the module first invokes Greenstone's code for ascertaining what collections exist on the system. For each Greenstone collection the module finds the collection's name and appends it as a child menu item to the main Greenstone menu item in Podzilla. This allows the user to choose a particular collection to work with.

In the example installation there are three collections (Figure 1a): *Advanced Beatles—Large*, *The complete works of Shakespeare*, and *AI Journal*. The menus are navigated using the iPod click-wheel, which moves the selected line up and down, with the selected menu item invoked using the iPod's central button. The user can travel back through the series of menus using the iPod's *menu* button just above the click-wheel. The whole process takes place very quickly: this kind of interaction is second nature for iPod users.

Interactive browsing

The next step was to map Greenstone's search and browsing functions into the iPod's interactive style, replacing the hyperlink navigation style implemented by the regular CGI version of Greenstone. In principle there is a clear and direct mapping of Greenstone's browsing capabilities into the iPod style of hierarchical menus. Greenstone collection designers can include a selection of browsing devices in their collection, like alphabetical lists and hierarchical browsers with arbitrary numbers of levels. All these browsing devices map naturally into the kind of hierarchy that iPod users are accustomed to traversing with the click-wheel.

In order to achieve this technically, Greenstone's access mechanism had to be recast as an iPod interface. Podzilla incorporates a graphical user interface toolkit specially developed for iPod-Linux, called TTK. This provides a stacked-window system, along with a high-level library including event handling and methods for input and graphical output, and forms an abstraction layer between the application and the lower-level graphics library.

Greenstone's implementation of browsing works by accessing metadata information in a database (we use GDBM, the Gnu database manager). We create menu items on the fly from the contents of the GDBM database as the user traverses the hierarchy, just as the regular CGI version of Greenstone does.

Figure 1 illustrates the process of browsing a collection. In Figure 1a the *Complete works of Shakespeare* collection is selected, and this happens to have only a *title* browser (Figure 1b)—as well as some search options. If the collection includes other browsers, such as *author* or *publisher*, these would appear on the menu in Figure 1b. The title browser is what in Greenstone is called an *AZ List*, for which the software automatically determines a suitable set of alphabetical ranges (overcoming the problem of empty ranges that occurs if all 26 letters are listed individually). Figure 1c shows the ranges for this collection. Selecting the first range leads to the titles beginning with the letters *A–I*, shown in Figure 1d. The user selects *All's Well That Ends Well* to get a list of pages in that particular document (Figure 1e). Selecting the first page, and scrolling down, we reach the speech by the *First Stranger* (Figure 1f). The TTK user interface toolkit automatically adds a scrollbar when necessary, as in Figures 1d and 1f: notice that the document has been scrolled down in Figure 1f.

Viewing documents

When a leaf of the browsing hierarchy is reached, Greenstone displays the document itself. To emulate this on the iPod, document launcher applications were written for text, image and audio media types, so that when a user browsed to a leaf node an appropriate action would be triggered. Text and image launcher applications made use of the TTK toolkit.

Surprisingly, it is not possible to access the iPod's native audio playing functionality from within iPod-Linux. Consequently presentation of audio documents was a challenging task. We were faced with the ironical situation of having a digital library system on a portable music player that could do everything but play audio! Eventually TTK was used in combination with the Linux digital signal processing file-mapped device, enabling the user to play an audio file, pause it, fast-forward and rewind it. We never managed to get volume control to work satisfactorily.

The display of textual documents is probably the weakest part of the Greenstone iPod implementation as far as practical deployment is concerned. We have been unable to locate a HTML renderer for Podzilla, and so HTML documents cannot be viewed properly. Our interim solution is to convert such documents to plain text before displaying them, but this eliminates all the formatting, hyperlinks, images, and so on. Greenstone collections do not have to use HTML documents—there is a standard plugin for plain text files—but in practice most of them do. Until a proper HTML renderer can be found, document display will remain unsatisfactory for most collections.

Searching

Full-text search has been a fundamental capability of the Greenstone digital library software from the very beginning. In

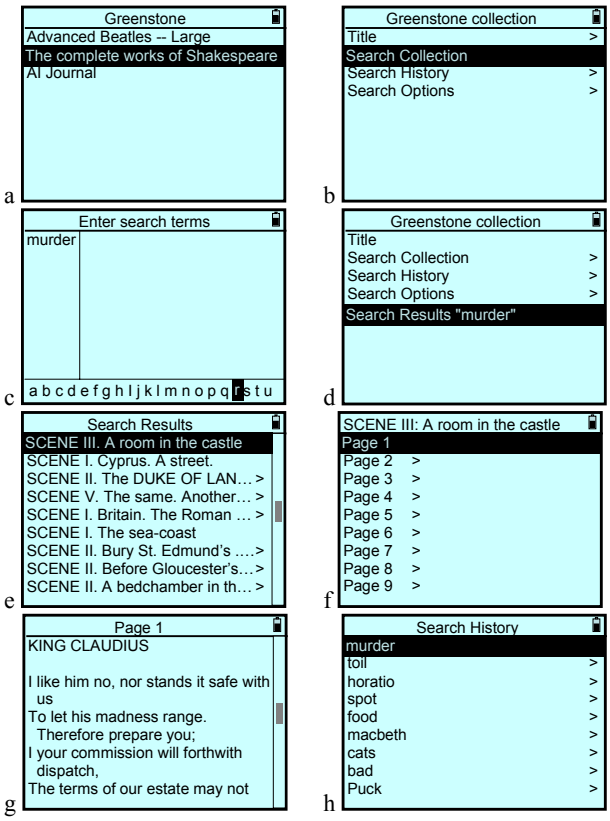

Figure 2 Searching Shakespeare for *murder*

the absence of any metadata, searching is the primary access mode for textual documents, and as far as the user is concerned it comes for free.

When implementing full-text search for the iPod, we came up against its minimal user interface. Although perfectly—beautifully!—designed for selecting and playing audio, with click-wheel and five "push" buttons (*play/pause*, *fast-forward* to end, *rewind* to beginning, *menu* and *enter*), there is no keyboard.

iPod-Linux comes with various text entry widgets: from a full range of characters displayed linearly on the screen (a-z, 0-9, punctuation, …) through which the click-wheel scrolls forward and backwards, to tapping out characters using Morse code. We did not attempt to come up with any significant innovation in this area, and decided to acquire the text for query terms using the linear selector, which is inevitably a rather frustrating experience for users.

Once acquired, initiating a query and displaying the search results is straightforward. However, text entry is so painful that we felt obliged to make past search results readily accessible to minimize the need for queries to be retyped. In Figure 2 the user is searching Shakespeare for *murder*. Having selected the collection (Figure 2a), they select *Search* (Figure 2b) and enter the search term (Figure 2c). Returning to the collection's main menu, they find that the search results have been added to the access list (Figure 2d). Selecting this item they see the results (Figure 2e) and can then access a particular document. Except for the actual text entry, all this is far easier to do than to describe in print. To iPod users, it feels perfectly natural.

There is a generic search history feature in Greenstone, but it does not associate past queries with particular collections. For the iPod, we decided to associate search history with individual collections, and make it persist across sessions (whereas the search results in Figure 2d will disappear if the collection were re-invoked). Figure 2h shows the history list. It is particularly useful because selection from a list incurs a far smaller human overhead on the iPod than retyping a query.

4.CONCLUSIONS

Other work on mobile digital libraries, e.g. [2, 3, 4], envisages a scenario whereby a personal digital assistant (PDA) connects, using wireless networking, to a central digital library server. In these systems the interface is web based, and the research involves tailoring it for a small screen (augmented, in some cases, with suitable data caching policies). Such work extends the existing web-based client-server model by adapting it for new kinds of client, a decision that is forced by the fact that most general-purpose PDAs do not begin to approach the storage capacity of centralized servers.

The vision we articulate in this paper is quite different: a complete, standalone, digital library, carried in the pocket on a mobile device. There is no need to access a network to retrieve content from a server. Indeed, in the future the portable device could actually act as a *server* of a vast amount of information. Our work has shown that this exciting possibility is now in reach.

We chose the iPod partly because of its high storage specifications but also because of its pervasiveness and popularity. The open source Greenstone digital library software forms the basis of the work. The system is not intended as a finished end product. At this stage it is experimental, and the value of the exercise has been to indicate the sorts of problems that arise when using such a device for digital library use, and illustrate the sorts of solutions that can be devised. The choice of Greenstone, therefore, was somewhat arbitrary: we made it principally because our own familiarity with the software allowed rapid development.

The result is by no means a finished product, suitable for end users. However, we find the idea of vast, fully-searchable, fully-browsable, digital library collections on a small portable device rather intriguing.

REFERENCES

[1] Witten, I.H. and Bainbridge, D. (2002) *How to build a digital library*. Morgan Kaufmann, San Francisco, CA.

[2] Buring, T. and Reiterer, H. (2005) "Zuiscat: querying and visualizing information spaces on personal digital assistants." In *Proc Int. Conf. on HCI with mobile devices and services*, pp. 129-136. ACM Press, New York, NY.

[3] Marsden, G., Cherry, R. and Haefele, A. (2002) "Small screen access to digital libraries." *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, pp. 786–787. ACM Press, New York, NY.

[4] Smeaton. A.F., Murphy, N., O'Conner, N.E., Marlow, S., Lee, H., McDonald, K., Browne, P. and Ye, J. (2001) "The Fischlar digital video system: a digital library of broadcast TV programmes." *Proc Joint Conf. on Digital Libraries*, pp. 312-313. ACM Press, New York, NY.