Accessing Speech Documents on Smartphones

Marcel-Cătălin Roşu
IBM T. J. Watson Research Center
19 Skyline Dr
Hawthorne, NY 10532
1 914 784 7242
rosu@us.ibm.com

ABSTRACT

This paper introduces *BBSearch*, which is an experimental system for exploring the challenges of ubiquitous access to recorded speech data. *BBSearch* applies information retrieval techniques to transcripts obtained by automatic speech recognition and it aims to provide a uniform user experience across platforms. To provide identical search functionality and document ranking, *BBSearch* applications use the same IR library for indexing and retrieval, namely Apache Lucene. For Java-enabled mobile platforms, *BBSearch* uses our J2ME Lucene port, called LuceneME.

This paper explores the resource requirements of LuceneME when used for Boolean searches and for supporting the podcast navigation GUI. On a BlackBerry smartphone, a diverse set of queries against a 70-hour corpus complete in less than 3 seconds and use less than 2MB of memory. The results of the evaluation validate our design and warrant expanding *BBSearch* to less capable cellphones, larger corpuses, or with more complex search capabilities.

Categories and Subject Descriptors

H.4.3 [Information Systems]: Communications Applications – information browsers; H.5.1 [Information Systems]: Multimedia Information Systems – audio input/output; H.5.2 [Information Systems]: User Interfaces – graphical user interfaces, input devices and strategies, interaction styles, natural language.

General Terms

Experimentation, Performance, Human Factors.

Keywords

Speech archive, search, smartphone, Lucene.

1. INTRODUCTION

In “As We May Think”, published in 1945, Vannevar Bush calls for a new relationship between what we call the knowledge worker and the sum of its knowledge [4]. Central to this relationship are the ‘memex’ and the human’s ability to access its storage by association. Decades later, PCs made the ‘memex’ device a reality. More recently, smartphones became the always-on/always-with device of the modern knowledge worker, with close to 120 million units shipped in 2007 and with the top vendors predicting growth rates above 50% for 2008.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MobiQuitous 2008, July 21–25, 2008, Dublin, Ireland.
Copyright © 2008 ICST ISBN 978-963-9799-27-1

The increasing computational and storage capabilities of smartphones made us explore the feasibility of a mobile ‘memex’ device. The emergence of podcasting as a tool for disseminating news and lectures, the advances in speech-to-text technologies, and the natural usage patterns of smartphones made us focus on enabling ubiquitous access to speech archives.

This paper introduces *BBSearch*, which is an experimental system designed to support ubiquitous access to recorded speech data, such as news podcasts, college lectures, or everyday life experiences [14]. To provide the best user experience for the available computing platform, *BBSearch* consists of several platform-specific implementations. All *BBSearch* applications aim at providing (1) the same search functionality, (2) identical document ranking algorithms, and (3) similar user interfaces. In addition, all applications use the same archive format, which enables archive sharing across platforms. Our focus is on PCs and Java-enabled smartphones, due to their high popularity among knowledge workers. On PCs, one application is browser-based while the other one is built as an extension of an existing podcast-management tool [3]. For Java-enabled smartphones, there is only one application, which runs on BlackBerry devices.

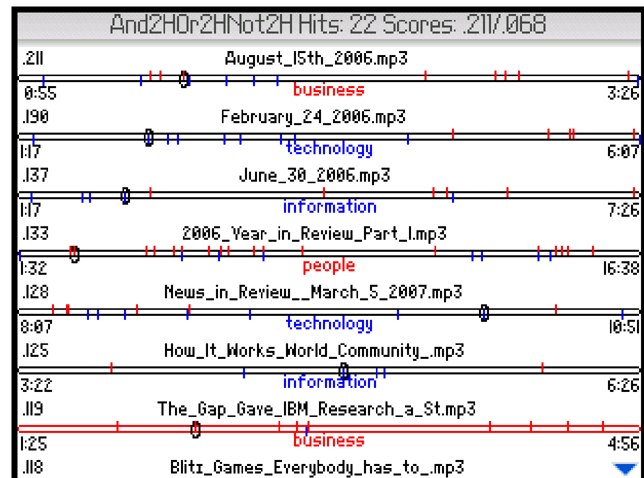

Figure 1. Search Results on BlackBerry 8800

BBSearch introduces a podcast-specific user interface for navigating search results (see Figure 1 and Figure 2), which is designed for easy and precise access to the desired recording(s). Although podcasts are easier to generate than text documents, they are significantly more difficult to access other than sequentially. The visualization of the search results is designed for easy global (within the result set) and local (within a podcast

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

timeline) navigation. Figure 1 and Figure 2 show the ranked search results (top 10) on a BlackBerry smartphone and PC, respectively. Each horizontal 'time line' represents a podcast, with the positions of the search terms marked by colored vertical bars.

The design of the smartphone user interface takes advantage of the trackball or four-way navigation key to enable one-handed operation. On the screen shown in Figure 1, vertical movements change the current podcast selection, which is highlighted with a different color. Horizontal movements chance the position of the oval-shaped cursor on the selected podcast. For each podcast, the search term associated with the marker closest to its cursor is displayed. The PC user interface uses only the mouse cursor for navigation; no podcast in the list is highlighted and podcasts are not associated a cursor. Mousing over a marker is necessary to displays the corresponding search term (see Figure 2). On both platforms, the markers associated with required (AND) and optional (OR) terms are displayed with different colors to enhance visibility.

The graphical representation of the search results, i.e., the markers representing the search terms and their absolute and relative positions on each podcast line, assist users in navigating within the result set. In addition, markers guide user navigation within a podcast: the user can hear a search term being uttered in context by clicking on the time line on or immediately before the corresponding marker.

BBSearch applies information retrieval (IR) techniques to transcriptions obtained by automatic speech recognition (ASR). The current prototype supports Boolean searches, expressed as terms combined with AND, OR and NOT operators. The transcripts used in this work are generated off-line, using transcription technology developed in the Human Language Technologies group in IBM Research.

On smartphones, the *BBSearch* application described in this paper performs searches locally: the archive and its index are stored in the cellphone's flash memory. The maximum size of the personal archive is determined by the size of the flash card and by the podcast format and compression factor; the indexing overhead is negligible, i.e., the index is about three orders of magnitude smaller than the data. Existing cards can accommodate a few hundred hours of speech data.

BBSearch departs from the traditional approach of accessing remote search capabilities via a mobile browser, as one of our goals is to run BBSearch applications on lower-end cellphones. We expect the capacity of affordable flash cards to increase much faster than the affordability of data-capable smartphones and associated data plans. For data-enabled smartphones, the traditional approach has the benefit of allowing access to much larger archives while the *BBSearch* approach is expected to allow faster access to information with lower battery consumption. To determine which approach is more desirable, a quantitative evaluation of the two approaches using realistic usage traces is necessary but such study is outside the scope of this paper.

For indexing and retrieval, *BBSearch* uses the same IR library across all platforms, namely Apache Lucene [1]. As a result, PC and smartphone applications support the same search functionality, rank results identically for consistent user experience (see Figure 1 and Figure 2), and can share archives (including their indexes).

The main Lucene distribution is written in Java 2 Standard Edition (J2SE). For the smartphone application, we create a new port to Java 2 Micro Edition (J2ME) for cellphones, called LuceneME. In addition, we extend Lucene with a transcript-specific analyzer and tokenizer, which store in the index the words *and* their timestamps.

The focus of the paper is to analyze in detail the resource requirements of the *BBSearch* implementation for the smartphone. This is motivated by the challenges of embedded and mobile platforms, such as cellphones; namely, their reduced computing resources when compared to PCs, and their limited, if any, capabilities for handling overload conditions. The data we collect on a BlackBerry helps us understand the scalability bounds of running IR applications on smartphones. We seek to answer questions like (1) what other types of J2ME-enabled phones can be used for *BBSearch*, (2) what is the maximum size of an archive that can be safely searched on a given cellphone, and (3) is it realistic to expand *BBSearch* with resource-intensive capabilities, such as wildcard, fuzzy and proximity searches. To the best of our knowledge, no similar systems have been analyzed from this perspective.

Our evaluation uses a 70hr corpus of enterprise podcasts and a collection of representative queries. All searches in our synthetic benchmark execute in less than three seconds and use less than 2MB of memory. Retrieving the timestamps of the search terms is much faster than the execution of the Boolean searches and it scales well with the index size and the number of hits.

In this paper, we do not formally evaluate the usability of the podcast navigation interface, but we briefly describe the feedback received on the browser-based interface. Also, we do not analyze the impact of ASR errors nor do we explore ways to compensate for them.

The following section provides some background on Apache Lucene and describes LuceneME. Section 3 describes the *BBSearch* applications. Section 4 describes the corpus used in the evaluation and Section 5 describes the results of the evaluation. Section 6 discusses related work. Section 7 summarizes our results and describes future extensions.

2. APACHE Lucene and LuceneME

Lucene started in 1997 as an IR library written in J2SE Java by Doug Cutting. Adopted by Apache in 2001, Lucene is now a much larger opensource project, which includes Lucene ports to C and C#/.Net. The widespread adoption of Lucene on PC platforms and the popularity of Java-enabled cellphones motivate the LuceneME port. The work described here uses Lucene 2.1.0.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

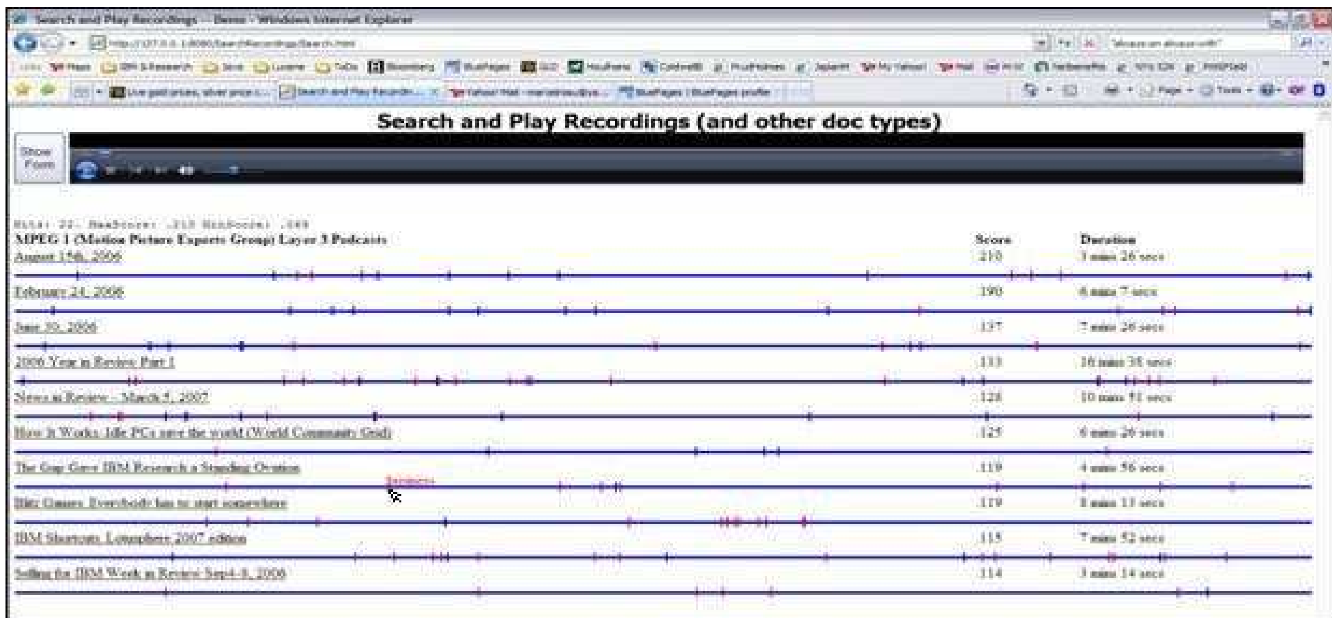

Figure 2. ***BBSearch*** **UI on PC**

2.1 Apache Lucene

Lucene uses a reverse index. A Lucene index consists of one or more segments. Each segment consists of multiple files, in the multifile index structure, or of one file, which encapsulates the above index files of the segment, in the compound index structure. Multisegment indexes and the compound index structure enable scaling Lucene applications to cluster-distributed indexes containing millions of documents. For faster access, Lucene caches in main memory part of the *term dictionary*, which is the most frequently accessed region of an index segment.

The logical view of an index is a collection of *Lucene documents (or ldocs)*. Each *ldoc* is identified by numerical ID and consists of a non-empty collection of *fields,* where a field is a (*name*, *value*) pair. For instance, (title, “Lucene in Action”), (ISBN, 1-932394-28-1), (body, “One of the key factors…..<<the rest of the words in the book>>”) could be fields in the *ldoc* representing [8].

Fields can be indexed, stored or both. Typically, fields like ‘body’ and ‘title’ are indexed and fields like ‘FilePath’ or ‘URL’ are only stored. Content searches use the indexed fields and yield the list of *ldocs* that satisfy the search query. Next, for each *ldoc* in the list, its stored field(s) are retrieved. Before being indexed, fields are analyzed.

Lucene analysis consists of converting text into tokens. More specifically, an analyzer extracts the words from the text discarding punctuation; some analyzers eliminate common words, perform lowercasing or more complex operations, such as stemming or lemmatization. Lucene includes an extensible collection of analyzers. Analyzers break a field value into a stream of *tokens*. After analysis, token values (words) and their positions in the document (sequence numbers) are stored in the index.

A Lucene query is a data structure that can be constructed by the application, using the Query API, or it can be generated by the QueryParser, from a string representation of the query. Parsing uses the same analyzer as indexing. A Lucene query is executed by the Lucene core. For instance, the smartphone application builds and executes its queries locally while PC applications compose their queries as strings and send them to the server for parsing and execution.

We extend Lucene with a transcript-specific analyzer and tokenizer, which allow us to replace in the index the word positions with the timestamps in the audio transcription (in tens of a second). As a result, timestamps can be retrieved using the SpanTermQuery Lucene API, which is optimized to take advantage of the index layout. For each word, its timestamps are stored sequentially, in increasing order of document ID and, within a document, in increasing timestamp value. Section 6 describes the performance benefits of this approach. The transcript-specific analyzer and tokenizer consist of about 1500 Java lines of code.

2.2 LuceneME

For LuceneME, our goals were (1) to port every Lucene feature that could possibly be useful on cellphones while preserving efficiency, (2) to preserve the index format, and (3) to minimize the number of changes to the retained code in order reduce the effort of leveraging future Lucene improvements. The porting effort is driven by three factors.

First, Lucene uses many J2SE features (classes, interfaces and exceptions) not included in the most popular Java runtime for cellphones, which is the Java Platform Micro Edition in the Connected Limited Device Configuration (CLDC) with the Mobile Information Device Profile (MIDP) or J2ME/MIDP[1]. For instance, the *ArrayList*, *HashMap* and *HashSet* classes and *List*, *Iterator* and *Set* interfaces are not available in J2ME/MIDP. In most cases, code using the missing features is rewritten to use those available. For the cases where the required code changes

[1] See [19] for a list of Java-enabled cellphones.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

were substantial, LuceneME adds minimal implementations of the missing features and the original code is left largely unchanged.

Second, the clone() method is missing from the Object class in J2ME/MIDP because CLDC uses KVM, which is a simplified Java VM designed for resource-constrained devices. Substituting for the lack of a clone() method resulted in many code changes, as our attempts to find an elegant and simple solution to this problem failed. For instance, access methods for the compound file structure frequently use object cloning to hide its underlying structure from the rest of Lucene, which is designed for the multifile index structure.

Third, we remove from Lucene features considered unnecessary on smartphones, such as handling parallel indexes or query parsing. We took a conservative approach to removing features, given that eliminating unnecessary classes or interfaces often requires changing the remaining code. Table 1 briefly summarizes the porting effort.

Table 1. LuceneME - Summary of Code Changes

Lucene 2.1.0 (lines of code)	33,800
Packages removed	1 (queryParser)
Files removed	4
Lines of code modified	600
Lines of code added	1500
LuceneME (lines of code)	31,400

3. *BBSearch*

BBSearch is designed as a *single system* spanning multiple platforms to provide the best experience on the computing platform available to the user. *BBSearch* aims to provide a uniform user experience across platforms, which includes the same search functionality, identical ranking of the result set, and similar user interfaces (subject to platform capabilities).

All but the last of these goals are achieved by using the same IR library for indexing and retrieval on all platforms, namely Lucene. In addition, the different *BBSearch* applications can use the same speech archive index, similar to the way media players on different platforms play exactly the same .mp3 files, because the Lucene ports preserve the index format. As a result, *BBSearch* users can move speech archives between devices using regular file copy operations.

BBSearch introduces a podcast-specific user interface for navigating search results. Namely, the result of a search is a ranked collection of timelines, one for each podcast, with the search term positions marked on the podcast timelines. The PC and smartphone UIs share this representation but handle input commands differently, as they use different input devices, i.e., mouse and trackball/4-way key, respectively. In addition, the smartphone UI is designed for one-handed operation when initiating previously stored searches, navigating search results, and playing the selected podcast.

A podcast is transcribed offline before it is added to the archive. Therefore, adding a podcast to a speech archive is similar to adding a text document to a text archive. Transcription time varies with the tool used, the size of its vocabulary, the desired accuracy and the computing resources of the transcription server. The tool used in this project runs on a high-end PC and it transcribes a podcast in about two times its duration.

A podcast transcript consists of a sequence of timestamped words, which is passed to the archive manager together with the location of the podcast. A new Lucene document (*ldoc*) is created for each podcast. The transcript, i.e., the sequence of timestamped words, is tokenized and the result is stored in the 'body' field of the new document. Other fields store the podcast location, title, author and duration. Typically, these values are extracted directly from the podcast; if the podcast is recorded in an .mp3 file, the values are extracted from its ID3 labels. *BBSearch* runs Boolean queries against the 'body' field. For each podcast in the result set, its title, author and duration are retrieved from its stored fields and displayed with the search results.

On PCs, *BBSearch* consists of a browser-based application and an extension of BlueBird, a Mozilla-based podcast management tool [3]. For smartphones, *BBSearch* consists of a BlackBerry application, which, except for its UI implementation, is J2ME/MIDP compatible.

The next section describes the PC applications in detail. The description reveals some of the challenges that a similar smartphone implementation would face: the need for a JavaScript enabled browser and for emulation of mouse pointer capabilities.

3.1. PC *BBSearch*

The two PC *BBSearch* applications access speech archives managed by an application server. The server handles commands for adding and removing podcasts, and for Boolean searches. The process of adding podcasts generates a new Lucene document, as previously described. The process of removing a podcast translates into a short sequence of Lucene API calls. In the following, we describe the search functionality.

The two PC applications are very similar. Both use an HTML form and JavaScript to input and process Boolean queries using three word lists, for AND, OR and NOT terms, respectively, and a positive number for the maximum size of the result set ('N'). The podcasts in the search result must include all the terms in the AND list, at least one from the OR list, and none from the NOT list. In the input form, AND and OR terms are displayed with red and blue, respectively; the same colors are used in the results screen to mark the positions of these terms.

The server is implemented as a Tomcat servlet, which uses Lucene and other open-source libraries to handle podcasts and other document formats. The server can also be run on the private desktop/laptop to keep the speech archive local. In this configuration, the applications access the server over the loopback interface.

Upon entering a query, the applications validate the input and construct the query expression string. The query, the AND and OR lists, and the value of N are sent to the server for processing.

The server parses the query string, which builds a Lucene data structure representing the query. Next, it executes the Boolean query by interpreting the data structure and it retains the top-N ranked podcasts. Lucene ranks the documents as they are being retrieved. For each podcast, the server retrieves the timestamps of

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

all the AND and OR terms received with the query string, using the SpanTermQuery API. For each podcast, the server also retrieves its title, duration, and length from the index. Using these values, the server constructs an HTML fragment and sends it back to the application. The receiver inserts the fragment in its DOM at a pre-set node.

Rendering the HTML fragment shows each podcast as a horizontal time line with the search terms displayed as vertical colored markers positioned in accordance with the terms' time offsets in the recording. When the mouse hovers over a marker, the text of the search term it represents becomes visible. Clicking on the timeline starts the podcast at the expected time offset: the browser-based application starts the Windows Media Player, while the BlueBird extension starts VLC [20], which is part of the BlueBird tool.

Internally, the HTML fragment represents each podcast as a separated section using a <DIV> tag. Within the section, the time line, the vertical markers and the terms are stacked in different layers. Markers and associated terms are positioned using coordinates computed from the timestamps retrieved from the index; initially, all terms are hidden. The visibility of the terms and the podcast play offset are controlled by JavaScript methods attached to the markers and the podcast time line, respectively.

Results are displayed in no more than a few seconds, when searching a 70hr podcast archive. To start a podcast at an arbitrary offset, one has to wait until enough of it is downloaded; this can be take up to a few minutes, depending on the offset, podcast bit rate and network conditions. While the missing segment downloads, the podcast plays from the beginning. Once downloaded, the play jumps at the desired offset. Downloaded podcasts are cached locally and subsequent accesses start playing immediately. Access to the desired offset is almost instantaneous when the server is hosted on the local machine.

A little more than a dozen people tried the browser-based application. The informal feedback was positive, with virtually everyone finding the interface intuitive and the application responsive. One suggestion was to provide more context by displaying several words surrounding the search term when hovering over the marker.

The same Tomcat servlet maintains a second archive, for various document types, such as PDF, Word, PowerPoint, XML, RTF, ASCII, etc. This second archive was used for an informal evaluation of the impact of transcription errors on search accuracy, as described in Section 4. The servlet implementation consists of about 1000 lines of Java.

3.2. Smartphone *BBSearch*

Currently, *BBSearch* runs only on Java-enabled smartphones. This platform was selected because of its popularity among knowledge workers, as all BlackBerry smartphones are Java-enabled, and because of the abundance of cheaper Java-enabled cellphone models, some of which having enough resources to host a *BBSearch* application.

The existing application architecture for smartphones manages only local archives. Extensions of this architecture with capabilities for searching remote speech archives are part of our future work.

The design of the smartphone application focuses on its usability and performance. This section describes the results related to the first focus area. Section 5 describes its performance.

The smartphone application enables one-handed operation by storing the most recent accessed podcasts in a play history and the most frequent term searches in a search repository; both history and repository are easy to navigate using only the trackball. The user interaction is structured around several overlapping screens. Figure 3 depicts the screen transition diagram.

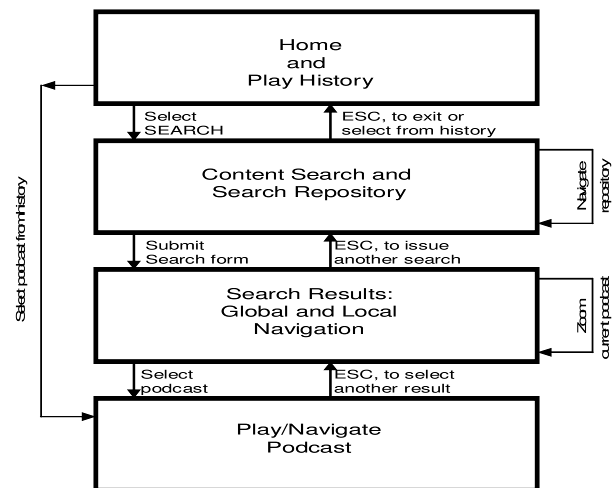

Figure 3. BlackBerry Application Screens

On the home screen, users can select between a podcast in the play history and starting a search. The play history records previously accessed podcasts; each podcast is described by the last played position and the positions of the search term(s) used to retrieve it.

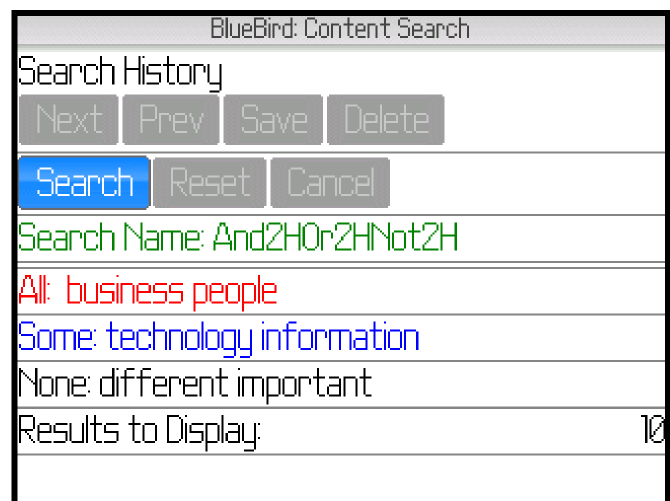

Figure 4. Search Screen

Figure 4 shows the "Content Search…" screen. The layout is designed for one-handed retrieval and execution of stored

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

searches using the trackball. For reuse, frequent searches are stored in a repository. The controls in the top row manage the search repository. When this screen pops up, the focus is set on the 'Next' button, to allow for a quick scan of the search repository. Clicking the 'Search' button initiates a search with the terms on the screen.

Search results are displayed in two stages. First, a transitory screen (not shown in Figure 3) displays the ranked set of podcast as a list of file names. This screen provides a 'quick' feedback to the user. Once term positions are retrieved, the "Search Results…" screen, which is shown in Figure 1, replaces the transitory screen. On this screen, the user can navigate between podcasts and zoom in and out of podcast segments using only the trackball.

Clicking on the podcast selection transitions to the media player screen partially shown in Figure 5 and sets the current play position to match the oval-shaped cursor in the previous screen. Figure 5 shows a media player that was enhanced to show the positions of the search terms on the progress indicator. Similar to the previous screen, the term closest to the current position is displayed (under the progress indicator).

Upon exiting the player screen, the user is prompted to save the current podcast and the associated context, which includes the current position and the search terms. Saved podcasts are added to the play history.

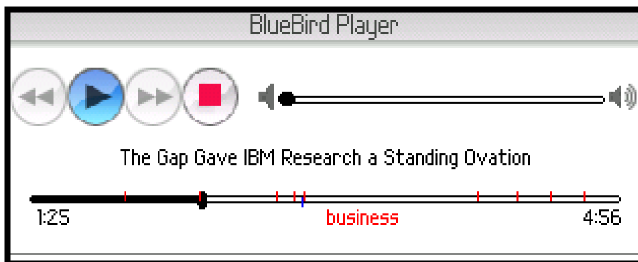

Figure 5. Enhanced Media Player (top ½ of screen)

To perform a search, the *BBSearch* application first composes and runs a BooleanQuery. The results of the query are displayed in the transitory screen. Next, a background thread issues a SpanTermQuery query for each search term and the retrieved timestamps are used to position the markers. After all these queries complete, the screen shown in Figure 1 pops up.

Except for the user-interface, the application can run on any J2ME/MIDP phone with the File Connection Optional Package (typically present). The Lucene port and an initial prototype were built using IBM's Device Developer (an Eclipse-based IDE), which was targeted at a generic J2ME/MIDP environment. For the next stage, which included the development of the user interface, we transitioned to RIM's JDE because the UI component uses many elements from the proprietary "net.rim.device.api.*" packages. RIM's JDE and its simulator for the 8800 were used for the final development and testing, and for capturing the screenshots. The implementation consists of about 5,800 lines of Java (not including the media player).

4. EVALUATION CORPUS

The evaluation of the smartphone implementation uses 350 podcasts from the IBM Media Library [7]. We identify the set of recordings with manual transcripts made between January 2006 and April 2007, remove the recordings in languages other than English and those sampled at less than 16KHz, and transcribe the oldest 350 recordings.

This corpus represents a little more than 70 hrs of podcasting and it requires 1.68GB of storage after conversion to mono sound. The podcasts are indexed in the order of their recording date. We create multiple indexes, comprising the first 25, 50, 75, … podcasts, respectively, and labeled them 'IndexNN', where 'NN' is the number of documents in the index (see Table 2). The second column shows the number of distinct terms in each index. The six most frequent terms considered representative for this corpus are: *'business'*, *'people'*, *'different'*, *'important'*, *'technology'* and *'information'*. The third column shows their ranks, which is the number of documents in the index that contain the term. The least frequent words (rank one) across indexes include *'ambient'*, *'disposable'*, and *'diffusion'*.

Early in the project, the same corpus was used for an informal evaluation of the impact of ASR errors on the quality of the search. Briefly, for each podcast, both the ASR and manual transcript were indexed in the speech and text archives, respectively, using one of the PC applications. A collection of queries were executed against both archives and results compared visually.

Table 2. Index Configurations

Config.	# of terms	Ranks of six Frequent terms	Index Size
Index25	4435	21, 23, 23, 20, 19, 18	120kB
Index50	6280	44, 44, 37, 36, 41, 38	201kB
Index75	7994	67, 65, 55, 53, 64, 59	297kB
Index100	9161	89, 86, 72, 68, 80, 73	377kB
Index125	10079	112, 108, 89, 83, 95, 85	449kB
Index150	11086	131, 129, 105, 100, 109, 100	533kB
Index175	12160	154, 145, 121, 119, 127, 115	629kB
Index200	12904	177, 161, 138, 132, 140, 131	710kB
Index225	13860	197, 178, 155, 150, 158, 149	798kB
Index250	14572	215, 194, 168, 169, 172, 167	895kB
Index275	15330	237, 214, 186, 190, 187, 185	989kB
Index300	15878	259, 235, 200, 207, 204, 204	1.06MB
Index325	16340	282, 251, 212, 222, 218, 221	1.13MB
Index350	16942	302, 272, 226, 237, 236, 236	1.21MB

5. EXPERIMENTAL EVALUATION

The evaluation uses an unlocked BlackBerry 8800 (EDGE) 4.2.1.72 (Platform 2.3.0.54) with CLDC-1.1 and MIDP-2.0. The 8800 has an Intel XScale 312 MHz CPU and a Sun JVM. Java tests report 56MB of total memory, which points to a 64MB RAM. The 8800's 64MB of flash is expanded with a 2GB microSD card, which stores the Lucene index and the 350 recordings.

The experiments measure separately the execution time and memory usage of Boolean queries and of the associated span

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

queries. Only the time to retrieve the data from the index and prepare it for display is reported; the screen manager overhead is not included.

Table 3. Benchmark Queries

Name	Query Expression
1. TermHA	+business
2. TermHB	+information
3. TermLA	+ambient
4. TermLB	+disposable
5. And2HA	+business +people
6. And2HB	+technology + information
7. And2L	+ambient +disposable
8. And3HA	+business +people +different
9. And3HB	+important +technology +information
10. And3L	+ambient +disposable +diffusion
11. Or2HA	+(business people)
12. Or2HB	+(technology information)
13. Or2L	+(ambient disposable)
14. Or3HA	+(business people different)
15. Or3HB	+(important technology information)
16. Or3L	+(ambient disposable diffusion)
17. And2HOr2H	+business +people +(technology information)
18. And3HOr3H	+business +people +different +(important technology information)
19. And2HOr2L	+business +people +(ambient disposable)
20. And3HOr3L	+business +people +different +(ambient disposable diffusion)
21. And2HOr2HNot2H	+business +people +(technology information) -different –important
22. And2HOr2HNot2L	+business +people +(technology information) -ambient –disposable

The 22 queries shown in Table 3 are executed in each of the 14 configurations in Table 2 and the top-10 ranked podcasts are displayed. The query collection, which is designed as a microbenchmark, uses search concepts introduced in TEXTURE [6]. The 'H's and 'L's in the query names designate that the query uses high- and low-rank terms, respectively. The digits and the logical operators in the query name designate how many terms are combined and their role in the query. For instance, 'And2HOr2HNot2H' searches for podcasts featuring two high-rank terms ('And2H'), one of two low-rank terms ('Or2H') and none of two high-rank terms ('Not2H'). 'A' and 'B' are used to differentiate between structurally identical queries. The 22 queries are stored in the search repository and executed in sequence. Plots display the average of three runs. In the following, we report some of the most relevant insights.

Execution times and memory usage vary with query type and the ranks of the terms used in the query. Figure 6 and Figure 7 show the execution times and memory usage for two simple, one-term Boolean queries, namely TermHA and TermLA. TermHA returns between 21 and 302 documents while TermLA consistently returns one document across all configurations. The execution times and memory overheads for TermHA's Boolean queries increase quickly for configurations with less than 200-250 documents; for larger indexes, they increase at a very slow rate and almost flatten even as the number of hits continues to increase linearly with the index size. For TermLA, the cost of finding 'ambient' in the term dictionary dominates the query overhead across all configurations.

When the ranks of the two query terms are both high and comparable, the overheads correlate well with the ranks of the two terms. For instance, the ratios between the execution times and memory usages of the TermHA and TermHB queries are close to the ratio between the ranks of the terms in the queries, i.e., 'business' and 'information', across all configurations.

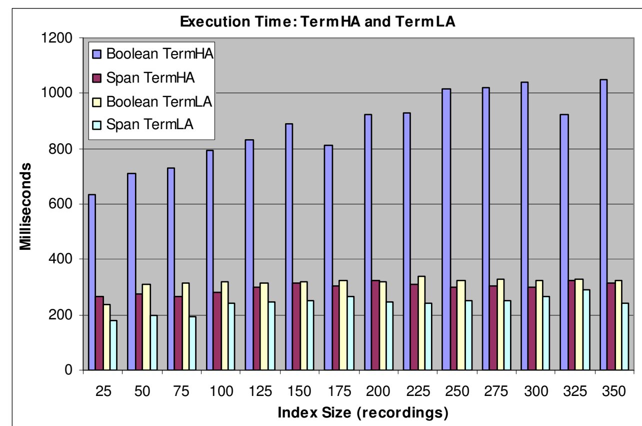

Figure 6. Execution Time: high vs. low rank terms

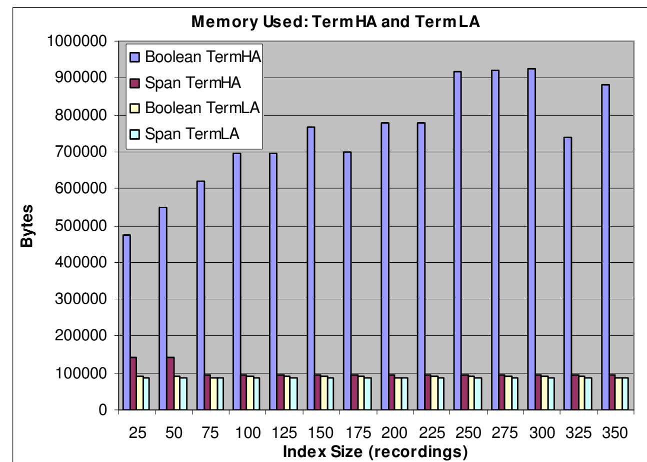

Figure 7. Memory Used: high vs. low rank terms

Results for TermLA and TermLB indicate that searches using a rare word are very fast, which is encouraging. Stored searches are expected to be used repetitively and they can be refined with rare

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

words to increase their selectivity and with common words to increase the expressivity of the result visualization.

The overhead of queries that yield an empty set is not negligible. For instance, And2L and And3L yield no podcasts and they take 250-300ms and 350-500ms, respectively, and use about 150kB and 350kB, respectively. Their execution times and memory usages are a higher than for TermLA, which returns one document across all configurations.

The execution time of Boolean queries using several high-rank terms and returning a large result set increases faster than their selectivity, i.e., the inverse of the cardinality of their result set. Their memory overheads correlate better with selectivity. For instance, Figure 8 and Figure 9 show the overheads of TermHA, And2HA and And3HA, which return between 21 and 302, 20 and 242, and 19 and 169 documents, respectively, across all configurations. These results teach us that it is preferable to refine Boolean queries with low-rank search terms.

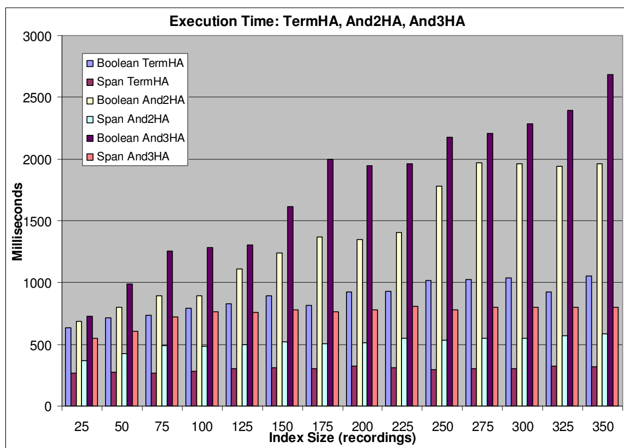

Figure 8. Execution Time: refining w/ high-rank terms

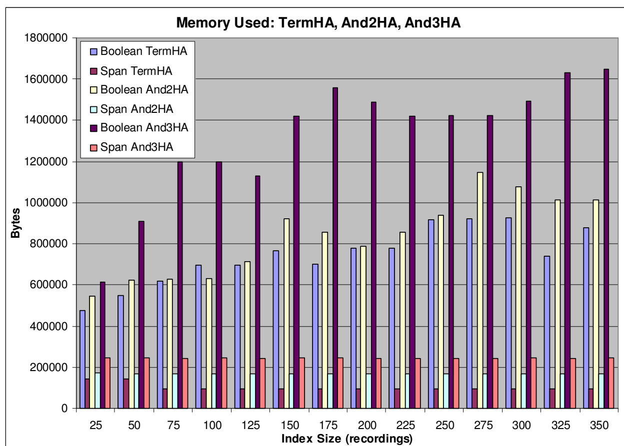

Figure 9. Memory Used: refining w/ high-rank terms

Using high-rank NOT terms in a query has a *positive* impact not only on the number of hits returned but also on its execution time. As expected, Lucene uses the negative terms early in the query evaluation process to prune documents from further processing. Figure 10 and Figure 11 show the case where the more complex query And2HOr2HNot2H requires fewer resources than simpler query And2HOr2H. And2HOr2HNot2H result size ranges from 1 to 22 while And2HOr2H result size ranges from 19 to 212 across all configurations.

Several Boolean queries exhibit a drop in memory usage and execution time for the 325-recording configuration. So far, our analysis indicates that this anomaly is more likely to be explained by the Lucene index organization than by an error in data collection or processing.

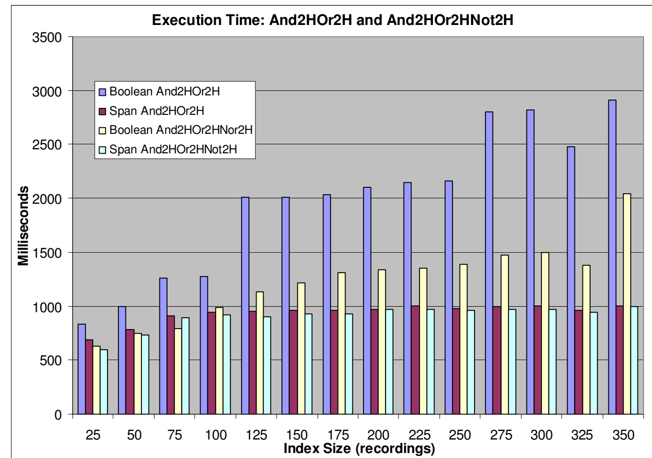

Figure 10. Execution Time: using negative terms

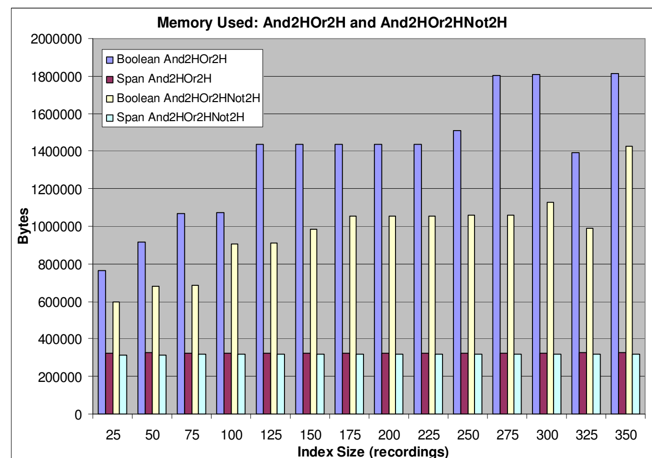

Figure 11. Memory Used: using negative terms

So far, we have only analyzed the overhead of Boolean queries. Resource requirements for SpanTermQueries, which retrieve word timestamps, are surprisingly low and constant across all configurations, despite the fact that more hits translate into more timestamps. This is explained by our approach of storing the timestamps in the index locations used for word positions and the use of the SpanTermQuery API, which is optimized to take advantage of the structure of the Lucene index.

First, for a given term, the SpanTermQuery retrieves the start of the index region where this term's timestamps are stored, for all of the documents in the index. The cost of this operation, which uses the term dictionary, increases logarithmically with the index size.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

Second, the SpanTermQuery scans this index region skipping over timestamps in documents not included in the result set. This results in close to constant overheads for the configurations used in this work. Therefore, the additional cost of representing term positions on the time line seem to be small, as the resources used for executing Boolean queries dominate those used by span queries.

Based on the results of our measurements, extending smartphones with extensive support for personal memories seems feasible. Although a 70 hrs archive is at the low end of what most of us would consider useful, our initial tests with larger archives and Lucene's success in handling much larger indexes on server platforms support our assertion. Overall, we are pleased with using *BBSearch* on the smartphone, as the search application feels very responsive. This is not unexpected, as we leverage the work done by the Lucene community on optimizing it.

Detailed modeling of query resource requirements is desirable but difficult given the complexity of the Lucene implementation. The structure of the term dictionary, which uses skip lists for faster access, appears to set a logarithmic upper bound on query execution time. Even without a detailed model of Lucene resource usage, our measurements suggest that it is safe to run *BBSearch* on collections several times larger on existing smartphones. Unfortunately, currently there are no smartphones with enough memory capacity to store large speech archives. Technology advances are expected to increase the capacity of memory cards or enable multiple card slots per cellphone.

6. RELATED WORK

Initial approaches to searching audio archives use text search methods on audio transcripts. It soon became apparent that a ranked set of audio files is less useful than a similar document set, due to the inherent sequential nature of the audio files that prevents them from being scanned quickly.

SCAN, for Spoken Content-based Audio Navigation [17], is the first system to propose a user interface design paradigm, called *What You See Is (Almost) What You Hear*, for local navigation of audio files. The SCAN approach is a multimodal method for accessing audio archives, which uses text search methods for retrieving a relevant list of recordings and their visual representations for local navigation. Recordings are divided in variable length segments using acoustic information and ASR is applied separately to each segment. The SCAN UI shows the ASR transcript and a histogram with a column for each segment of the story. Search terms found in each segment are displayed as stacked, variable-height rectangles; this representation, called TileBars, was introduced in [9] for text documents. Jotmail [15] and later SCANMail [16], focus on providing voicemail users with a visual representation of their archive. Finally, the term "strategic fixation" is defined as the "visual scan of text to focus on regions of interest" in [18], which is also summary of the authors' experiences with building Jotmail and SCANMail.

BBSearch is designed to support the search and navigation of audio archives as well, but it is intended for ubiquitous access to a larger collection of recordings than voicemail. In contrast to these systems, the *BBSearch* UI shows the visual representation of *several* elements in the ranked set *at the same time*, therefore assisting in global navigation, as well. For local navigation, *BBSearch* allows for higher precision in locating relevant utterances than previous systems.

The influence of ASR accuracy on user experience is determined to be linear and only transcripts with word error rates less than 25% are usable in searching Webcast archives [11]. The impact of ASR accuracy on the effectiveness of the SCAN system and its user interface is analyzed in [12]. *BBSearch* does not address these topics.

On personal devices, handling speech-as-data is even more challenging due to the inherent limitations of these devices, which leads to removing resource-intensive features from the mobile version of an application. For instance, the UIs of the two mobile implementations of SCANMail display voicemail headers with little extra information [18]. We found only one graphical UI for random-access to speech records on personal devices [13]. A speech record is divided into variable length chunks based on pauses in the recording; on average, chunks are five seconds long and each chunk is displayed as a continuous horizontal line, one after the other, like words in a paragraph. The player controls allow for direct navigation from one chunk to another. *BBSearch* allows for a more precise navigation based on search terms.

Currently, there is significant research interest in designing *personal* information devices or management systems, which handle email, Web page history and images in addition to recorded conversations or lectures. These systems are intended to support our memories and recent studies identify that *device efficiency* [10] and *fast response* [5] are the most desired characteristics.

There is a strong motivation for personal information devices with very large storage, and existing technologies support their design and implementation. The convenience of a *single always-on*, *always-with* and *connected* device points to the cellular phone as the preferred platform to be expanded with support for indexing/searching large amounts of personal information. For cellphones, audio recordings are the preferred information medium, mainly because of the small size of their displays. Our work aims at advancing the understanding of the feasibility of this paradigm.

7. CONCLUSIONS and FUTURE WORK

This paper describes *BBSearch*, a system for ubiquitous search of speech archives. *BBSearch* introduces a podcast-specific user interface for global and local navigation of search results, and it includes PC and smartphone applications for managing speech archives. The latter uses LuceneME, our Lucene port to J2ME.

The paper focuses on the resource requirements of LuceneME when used for searching smartphone-resident indexes. The results of our experiments show that searches complete in less than a few seconds and use only a small fraction of the available memory. We learned several lessons from the experimental evaluation. For instance, the cost of retrieving the timestamps used by the graphical user interface is lower than initially expected.

In our future work, we plan to explore how *BBSearch* can use other types of Lucene queries, such as proximity and fuzzy queries, against errorful transcripts and how to run them efficiently on smartphones. We are also looking at how to provide

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635

user feedback during query specification towards avoiding queries with high resource usage and low selectivity. We are also looking at how the existing smartphone application design can be extended with capabilities for searching remote archives and integrating the results of local and remote searches.

8. ACKNOWLEDGEMENTS

The media player shown in Figure 5 was written by Dan Coffman. Brian Kingsbury was instrumental in obtaining the ASR transcripts. Comments from the anonymous reviewers helped improve the presentation.

9. REFERENCES

1. Apache Lucene, http://lucene.apache.org.
2. E. Adar, D. Karger and L. Stein. Haystack: Per-User Information Environments. In CIKM99, 413-422, 1999.
3. W. Bodin, A. Grizzaffi. Enterprise Library Management for Digital Media with Dynamic Media Synthesis. In ISWPC07, 373-377, 2007.
4. V. Bush. "As We May Think", Atlantic Monthly, July 1945, www.theatlantic.com/doc/194507/bush
5. S. Dumais, E. Cutrell, J.Cadiz, G. Jancke, R. Sarin, D. Robbins. Stuff I've Seen: A System for Personal Information Retrieval and Re-Use. In SIGIR03, 72-79, 2003.
6. V. Ercegovac, D. DeWitt and R. Ramakrishnan. The TEXTURE Benchmark: Measuring Performance of Text Queries on a Relational DBMS VLDB05, 313-324, 2005.
7. G. Faulkner. Podcasting and Social Media at IBM. http://gfaulkner.wordpress.com/2007/11/05/social-media-at-ibm-focus-on-podcasting/.
8. O. Gospodnetic and E. Hatcher. Lucene In Action. Manning Publications 2005.
9. M. Hearst. TileBars: Visualization of Term Distribution Information in Full Text Information Access. In CHI95, 59-66, 1995.
10. V. Kalnikaite and S. Whittaker. Software or Wetware? Discovering When and Why People Use Digital Prosthetic Memory. In CHI07, 71-80, 2007 (Best Paper).
11. C. Munteanu, R. Baecker, G. Penn, E. Toms and D. James. The Effect of Speech Recognition Accuracy rates on the Usefulness and Usability of Webcast Archives. In CHI06, 493-502, 2006.
12. L. Stark, S. Whittaker and J. Hirschberg. ASR Satisficing: The Effects of ASR accuracy on Speech Retrieval. In International Conference on Spoken Language Processing, 1069-1072, 2000.
13. R. Tucker, M. Hickey and N. Haddock. Speech-as-data technologies for personal information devices. In Pers Ubiquit Comput, 7:22-29, 2003.
14. S. Vemuri, What Was I Thinking?, http://web.media.mit.edu/~vemuri/wwit/
15. S. Whittaker, R. Davis, J. Hirschberg and U. Muller. Jotmail: a voicemail interface that enables you to see what was said. In CHI00, 89-96, 2000.
16. S. Whittaker, J. Hirschberg, B. Amento, L. Stark, M. Bacchiani, P. Isenhour, L. Stead, G. Zamchick, and A. Rosenberg. SCANMail: a voicemail interface that makes speech browsable, readable and searchable. In CHI02, 275-280, 2002.
17. S. Whittaker, J. Hirschberg, J. Choi, D. Hindle, F. Pereira, and A. Singhal. SCAN: designing and evaluating user interfaces to support retrieval from speech activities. In SIGIR99, 26-33, 1999.
18. S. Whittaker and J. Hirschberg. Accessing Speech Data Using Strategic Fixation. In Computer Speech and Language 21(2), 296-324, 2006.
19. The Java ME Device Table, http://developers.sun.com/mobility/device/device.
20. VLC Media Player, www.videolan.org

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3635
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3635