Lifetrak: Music In Tune With Your Life

Sasank Reddy
NESL
Department of Electrical Engineering
University of California, Los Angeles
sasank@ee.ucla.edu

Jeff Mascia
NESL
Department of Electrical Engineering
University of California, Los Angeles
jmascia@ucla.edu

ABSTRACT

Advances in sensing technology and wider availability of network services is beckoning the use of context-awareness in ubiquitous computing applications. One region in which these technologies can play a major role is in the area of entertainment. Particularly, context-awareness can be used to provide higher quality interaction between humans and the media they are interacting with. We propose a music player, Lifetrak, that is in tune with a person's life by using a context-sensitive music engine to drive what music is played. This context engine is influenced by (i) the location of the user, (ii) the time of operation, (iii) the velocity of the user, and (iv) urban environment information such as traffic, weather, and sound modalities. Furthermore, we adjust the context engine by implementing a learning model that is based on user feedback on whether a certain song is appropriate for a particular context. Also, we introduce the idea of a context equalizer that adjusts how much a certain sensing modality affects what song is chosen. Since the music player will be implemented on a mobile device, there is a strong focus on creating a user interface that can be manipulated by users on the go. The goal of Lifetrak is to liberate a user from having to consciously specify the music that they want to play. Instead, Lifetrak intends to create a music experience for the user that is in rhythm with themselves and the space they reside in.

Categories and Subject Descriptors

H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Systems*

General Terms

Algorithms, Design, Human Factors

Keywords

Context, Entertainment, Mobile, Music, Sensors

1. INTRODUCTION

One of the most popular media for personal electronic entertainment is the portable music player. These devices have become even more popular in recent years due to introduction of convenient distribution platforms for music and also the availability of devices that can store a large collection of music in a package that can sit in the palm of one's hand [1], [2]. Most usage scenarios associated with these mobile music devices involve creating custom playlists for different situations. Often, these playlists correspond to situations that can be characterized by attributes such as the user's mood, location, or the time of day.

Although pre-defined playlists offer a good method for organizing music, the user must manually change playlists to match the current environment or mood. This mode of operation is undesirable. [3] shows that mobile environments, especially ones that surround an urban setting, are very dynamic and unpredictable so frequent changes in music is required to match the user's preferences. Furthermore, interacting with most mobile devices is a tedious task because user input requires explicit attention and time. This concept accounts for the sustained popularity of FM radio; one simply turns it on and listens. Radio also highlights the problems with random music selection: songs can be out of touch with the user's mood or environment.

These existing challenges in the portable music space along with user desire for dynamic music selection serve as the motivation for Lifetrak. The essential idea behind Lifetrak is that several sensing modalities influence the music that is played. Essentially, Lifetrak enables a context-aware playlist that automatically chooses music in real-time based upon the location, the pace of movement, the current time, and various other phenomena in the user's environment. Furthermore, Lifetrak uses a simple learning mechanism to adjust the ratings of songs for a particular context based on user's feedback when a song is being played.

An important aspect of Lifetrak that must be emphasized is the manner by which music selection actually takes place. The system does not autonomously analyze songs and connect them to corresponding contexts. This could be a dangerous practice because different users may prefer to hear opposite types of music in the same context. Instead, Lifetrak relies on the user to tag the song database with the basic context constructs. Lifetrak then figures out the user's current context and ranks the song database according to this information. The result is a list of the most appropriate songs for that particular situation.

The remainder of the paper is organized as follows. We discuss previous work including current music players that exist, how context awareness is currently used in ubiquitous computing, and work related to user interfaces in a mobile setting in Section 2. Motivational usage scenarios for Lifetrak are provided in Section 3. Section 4 contains a general overview of the system architecture. Details regarding the context engine and also the actual hardware used for the player are described in Section 5. Section 6 provides information regarding the actual music player, in terms of design, user interface, and features. Guidelines for future work and the conclusion for the paper are given in Sections 7 and 8. Finally, Section 9 contains acknowledgments. Also, we have an Appendix that has examples of various XML files used in Lifetrak.

2. PREVIOUS WORK

2.1 Music Players

There are numerous mobile music players that currently exist ranging from the Apple iPod to the Sony Walkman MP3 Player [2], [4]. These players usually have the ability to create custom playlists that the user can select. But in terms of dynamic change, the only functionality they provide is the ability to shuffle songs in a certain playlist or a library. In terms of music players that adjust based on sensing, [5] tries to match a user's speed to songs that have corresponding similar tempos. [6] takes the tempo matching technique further by introducing physiological data, such as the user's heart rate. In essence, most smart playlist music players just do simple classification and do not utilize the array of sensing modalities available in Lifetrak. The idea of tagging music is nothing new. [7] and [8] talk about how tags can be introduced for music and collaborative tagging techniques. In fact, [9] has a web interface to upload songs and tag them arbitrarily. Queries regarding the tags and songs can be made using a web service interface. LastFM enables an unlimited amount of tags while Lifetrak confines tags to the contexts that it uses in its music selection algorithm.

2.2 Context Awareness

Context-aware computing has come to the forefront in recent years especially in the ubiquitous computing community. But most of the work in this area leverages only a user's location [10], [11], what a user actually inputs or attests to [12], or historical pattern matching techniques [13]. This is mainly because obtaining meaningful contexts is a very complex and hard process. There have been many studies on what contexts should be used and how they should be used. For instance, the idea of using social interactions and dynamics for context generation has been suggested by [14]. Also, [3] reviews the complexity of obtaining useful contexts in an urban environment where there are many external factors that change often and are unpredictable. Furthermore, context classification involves many different sensing modalities and typically aggregating and making a decision on these modalities requires long wait periods and high overhead computing capabilities. Lifetrak takes an approach of obtaining context with sensors that will be available on a typical mobile device in the same class as a smart-PDA phone. This enables us to focus on fine tuning the actual classification of the contexts and using the resources that are available to the fullest.

2.3 Mobile User Interfaces

User interface design is very important in the realm of mobile computing. Typically, mobile devices are fairly small and have a small screen to interact with. Furthermore, there is not a full keyboard or mouse setup that comes with these devices. Taking the approach of porting existing workstation user interface environments and directly implementing them on a mobile devices is not appropriate [15]. Furthermore, requiring both hand and visual attention to operate a device does not fit well in most situations where mobile devices are being used due to the fact that the environment that the user exists in requires multitasking [3]. Finally, [16] suggests that interfaces generally have to be made less complicated and quick to enact. Lifetrak was designed with having a mobile user that is engaged in multiple actions in mind. The user interface for the music player is simple but effective in conveying the necessary information, and interaction with the device is made easy by having multiple input methods ranging from push buttons to a touch screen.

3. USAGE SCENARIOS

To illustrate the need for Lifetrak, we attempt to describe the listening experience of a stereotypical portable music user. The massive storage capacity of today's portable music players offer access to a wide variety of music on demand. However, such extensive availability presents somewhat of a quandary for music selection. The laborious process of navigating one's musical library for the next song to be played can detract from the enjoyment of simply listening to music. Furthermore, music listening is often a background process. Users listen to their portable music players while exercising, reading, working, driving, etc. Hence, time spent selecting music only distracts users from the task at hand. Although one may avoid this problem by choosing songs at random, it depersonalizes the listening experience. We believe that different situations elicit different emotions for different people and consequently call for different music. Our goal for Lifetrak is to reduce user input to the operational loop while maximizing user situational enjoyment.

To see how this works, let's observe a hypothetical Lifetrak user. Upon importing his or her music collection to Lifetrak, the user labels songs, albums, or artists, with predefined context tags representing the situations in which the user desires to hear them. Admittedly tagging is not a trivial task, but since users often commit the effort to repeatedly make playlists, we believe they will commit to a one-time tagging process. During daily usage, Lifetrak uses various sensing modalities to monitor context and select songs based upon their tags. For instance, depending on the preferences of our hypothetical user Lifetrak may play rock music when out for an afternoon jog, and classical music when trying to study in a noisy cafeteria. Furthermore Lifetrak may play calm, soothing music during stressful situations such as hurrying when late to class or waiting in traffic on the way to the beach. These examples represent only a fraction of the situations in which Lifetrak can improve the user's daily experience.

4. SYSTEM OVERVIEW

4.1 Hardware

In order to implement Lifetrak, the Nokia 770 Internet

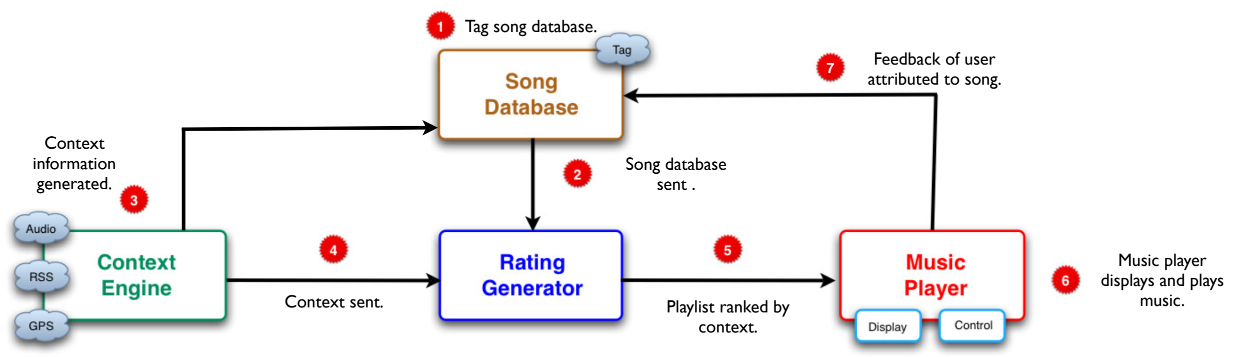


Figure 1: Lifetrak operation overview.

Tablet was used as the hardware platform [17]. This device provides an ideal platform for our application for a few reasons. First, the Nokia 770 provides WiFi connectivity and the ability to interface to a cellular network through the use of a phone that is attached via Bluetooth. Also, Bluetooth connectivity enables a Holux GPS unit to be available for use with the device. The Nokia 770 has an internal microphone for acquiring audio samples of the environment. The actual device will be featured in later sections. We envision that in the future Lifetrak can be ported to a variety of embedded computing systems platforms including smart-phones, standard MP3 players such as iPod, and even satellite radio receivers.

4.2 Software

There are basically four main components involved in the software architecture for Lifetrak. First is the user space document that contains a list of all songs and the contexts that apply for each song. Then there is the context engine that actually gets information from various sensing modalities and categorizes them into specific tags. This process is also referred to as getting the current user context. Also, there is the rating generator which takes the current context and the user defined song database, which contains the individual context tags for each song, and then generates a ranked playlist. The context engine and rating generator modules were done as Python modules due to the fact that parsing for XML and RSS can be done easily and also access various sensing modalities is made simple. Finally, there is the music player which provides an interface for the end user. Figure 1 contains all the entities in the architecture along with the operations performed by each component.

In order to illustrate how the Lifetrak system works in general, the typical operations will be described below.

1. **Songs Tagged**

   One of the most important aspects of Lifetrak that differentiates it from other context-using music players is that the system does not try to match the sensing modalities to music by analyzing the song. Instead, it relies on the user providing context tags that represent when the song should be played. For instance, some typical tags might include ”morning”, ”90024” (zip-code), or ”sunny”. A complete list of tags that can be applied are shown in Figure 7. The song database is an XML file which is currently modified through a external viewer. Future work in this area include creating a song database manager that is able to tag songs with a graphical user interface and provide searching capabilities.

2. **Song Information Transfered**

   Once the song information is actually tagged appropriately with relevant context information, then the information is made available to the rating generator.

3. **Contexts Obtained**

   Context information for the user space is obtained through various sources. A GPS unit is used to get the location and the speed of the user. Several syndication (RSS) feeds are obtained as well to get weather and traffic information for a particular location. Finally, a microphone is used to get the decibel level of an environment.

4. **Context Information Transfered**

   The context information is also sent to the rating generator for further processing. Currently the context information is obtained at a default period of every one minute. But this is configurable.

5. **Songs Rated and Playlist Created**

   The rating generator takes the song database, which has tags associated with each song corresponding to the context that they should be played, and compares it with the current context and then ranks the songs. The ranking is then used to create a playlist XML file that can be read by the music player.

   The ratings are generated by comparing the current context to each of the individual song's context information. Basically, if the current context matches one of the specific tags for a song in the database, then that song's rating is increased by the amount specified by that tag. The ranking is found for all songs in the database and then the songs are played from highest ranking to lowest. More detailed information about the rating process can be found in Section 5.

6. **Music Played and Controlled**

   The music player acts as the GUI for music playback. It has several methods, soft buttons in the GUI and the keypad buttons, to control the playback of the actual songs. It updates the screen with the current context information and any changes made to the playlist rankings.

7. **Feedback Information Transferred**

   Finally, feedback is gathered from the user in order to correct whether a certain song should be played in a context or not. This type of feedback is obtained by having a method to rate a song on whether one likes the song or not and also by changing how much each context affects the rankings as a whole. This is referred to as the ”love it/hate it” functionality and the context equalizer. More information about this feedback system is given in Section 5.

5. CONTEXT

One of the key features differentiating Lifetrak from other forms of mobile entertainment is its use of context information to choose what songs to play for the user. The following sections describe details about the different contexts in terms of categories, context management scheme, and also how user feedback is incorporated into the context engine.

5.1 Types of Context

5.1.1 Space

Space, or location, can have a significant influence on music selection. For example, urban environments may evoke different emotions than do rural environments. This same contrast can exist for beaches and mountains, or other geographic characteristics. Furthermore, spatial knowledge is necessary to retrieve remotely sensed contextual information such as the weather. The granularity of spatial description provided to the user is an important design choice. Space can be represented by GPS coordinates, a polygon bound on GPS coordinates, a zip code area, or even higher levels of abstraction. After polling typical users and weighing

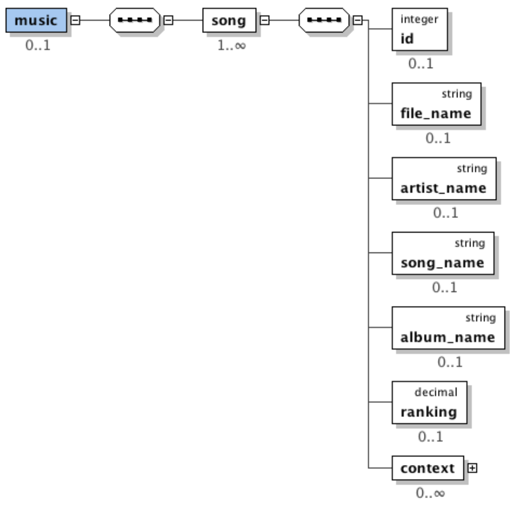


Figure 2: Model representation of song database XML.

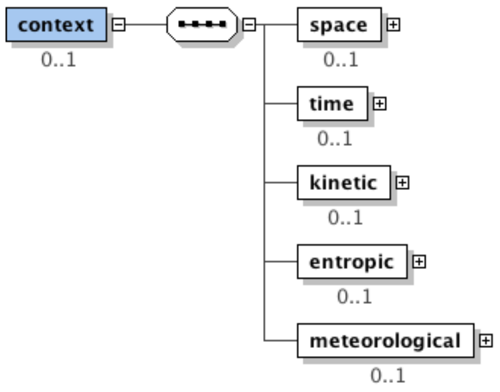


Figure 3: Context information XML schema.

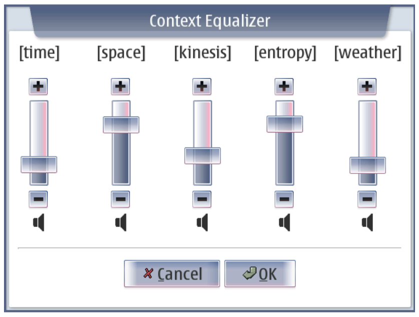
the options, we concluded that high level descriptions work better. Since general descriptions such as beach, school, or work, are hard to verify, we chose zip codes. The user is free to associate zip codes with personal landmarks or geographic descriptions. Lifetrak transforms the user’s GPS location to a corresponding zip code by performing a simple closest distance approximation using data obtained from the US Census. Although this translation is not necessarily a one to one mapping, since zip code boundaries vary and are not uniform, we believe this is a good enough approximation in most cases. When Lifetrak is unable to receive adequate GPS signals, it uses the last known GPS coordinates to determine its zip code. Lifetrak’s space context also includes tags for whether the user is “outside” (receiving a GPS signal) or “inside” (not receiving a GPS signal).

5.1.2 Time

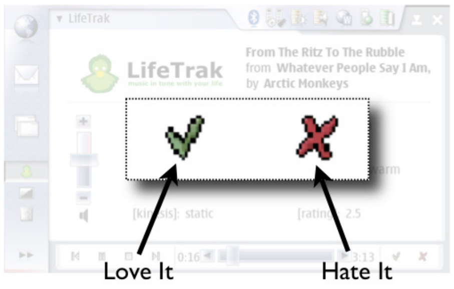
Time is another context that directly affects the music to which people want to listen. Users often prefer listening to certain types of music at different periods of the day. In addition, users may associate activities with certain times during the day, such as taking a morning run, coming back from work in the evening, or going out to lunch. Often these activities can be described in terms of time. Lifetrak abstracts time into predefined states for the user. First, a typical day is divided into periods. These tags include “morning” “afternoon”, “evening”, and “night”. Since a user’s perception of these periods can vary, the time boundaries defining these states are configurable. In addition, Lifetrak represents the time context in terms of the day of the week. To measure time, Lifetrak uses the internal clock of the Nokia 770 rather than relying on GPS because GPS is not always available. Ideally the Nokia’s clock should be periodically synchronized to maintain long-term accuracy.

5.1.3 Kinetic

The speed of the user is another context that helps Lifetrak decide what music to play. For example, a particular user may prefer motivational music when running for exercise and peaceful music on a potentially stressful drive home from work. Lifetrak relies on speed information provided by the GPS unit. Although GPS accuracy is adequate for determining different modes of movement, kinetic information is limited by the availability of GPS signals. In the fu-



Context Equalizer



Song Feedback

Figure 4: User feedback operations.

ture, we envision extending Lifetrak's capabilities to indoor settings by adding accelerometers and gyroscopes. Kinesis is currently divided into four distinct categories: "static", "walking", "running", and "driving". The speed bounds on these categories were established by analyzing a sample set of users, but they can be customized for each individual. To provide more consistent measurements, Lifetrak calculates the user's average speed over a period of time.

5.1.4 Entropic

The state of the environment is represented by the entropy context. This value is obtained in two different ways. If the user is not driving, Lifetrak calculates the mean decibel level from a five second audio sample. The decibel level gives an indication of the busyness of the user's environment, which Lifetrak categorizes as either "calm", "moderate", or "chaotic". For example, a user may prefer to listen to different music when standing on a busy street corner than when studying in a quiet library. Conversely, the microphone is not sampled if the user is driving. Instead, Lifetrak acquires the user's local traffic conditions from an internet RSS feed and categorizes them into the aforementioned tags. The default values of the entropy categories were obtained by analyzing decibel and traffic patterns for a city setting, but as always they may be customized by the user.

5.1.5 Meteorological

Weather plays an important role music selection because weather can dramatically affect one's mood. Most commonly, a user may associate different music with sunny days than with rainy days. Lifetrak pulls weather information from RSS feeds and divides the meteorological context into tags representing the outdoor temperature and condition. The temperature is categorized as either "frigid", "cold", "temperate", "warm", or "hot". The condition is categorized in typical meteorological terms as described by the source of the RSS feed, currently Yahoo.com.

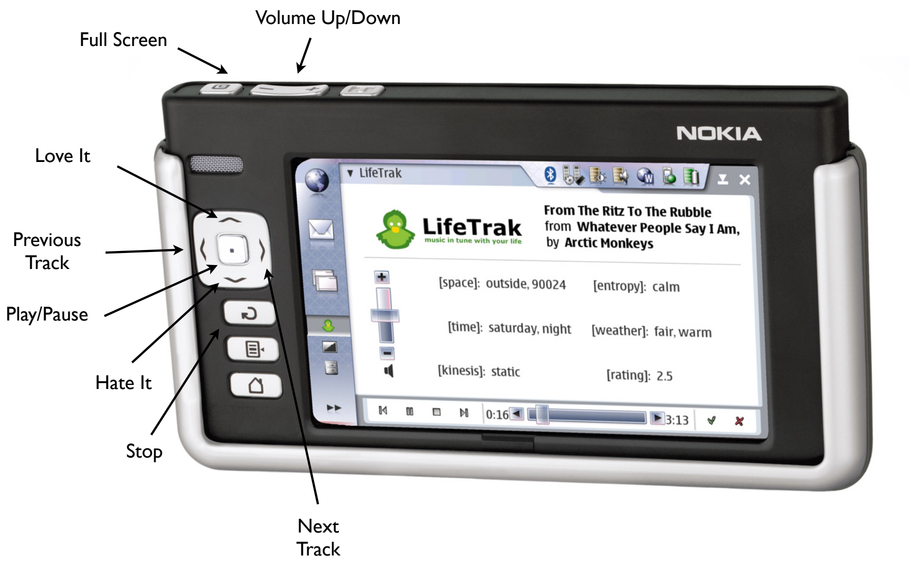
5.2 Song Rating Algorithm

The method which Lifetrak uses to determine song ratings for a particular context is an important feature that distinguishes it from other music players. When music selection utilizes sensing, often a song is chosen by analyzing patterns in the song itself and then correlating them to the sensed modality. However in the case of Lifetrak, the user's personal preference is at the forefront. Lifetrak's methodology, in particular the song rating algorithm, is described below.

Lifetrak's song database is an XML file containing metadata and context preferences for all songs in the user's music library. Figure 2 shows a model representation of the song database XML file. Each song entry in the database includes a field for every possible tag the user can attach to a song. In our current implementation, Lifetrak provides the user with a finite set of possible tags from which to choose; each tag corresponds to a specific context state. Hence the available tags for the kinetic context are "static", "walking", "running", and "driving". The user can choose to attach any subset of these tags, and the tags associated with the other contexts, to each song. For each tag the user attaches to a song, Lifetrak assigns a value of one in that tag's field. For each tag not selected by the user, Lifetrak assigns a value of zero in that tag's field. Essentially the value in a tag's field represents the user's desire to hear that song in the tag's context state. These values can later be altered by user feedback and can be any real number from zero to one.

Lifetrak uses a similar method to store information about the user's current context. After Lifetrak obtains data for each context category (space, time, etc.) via its various sensing modalities, it then classifies the state of each context category ("90024", "morning", etc.). Lifetrak assigns a value of one to each of the current context states and a value of zero to all other possible states.

Lifetrak then uses the tag values from the song database and the values of the current context states to calculate a rating for every song in the database. This rating represents the users overall desire to hear a certain song in the current context. The algorithm for calculating the song rating is not complex. For a song, Lifetrak multiplies the value in each tag field, by the corresponding state value in the current context. The song rating is the sum of these multiplications. Alternatively, we can also describe this computation as follows. Suppose we consider the set of all tag values in a song entry to be a vector. Note that the set of all possible states in the current context composes a vector of identical dimension. Thus a song's rating is merely the dot product of the song's tag vector and the current context's state vector. For example, suppose a user attaches the following tags to a song: "saturday", "sunday", "driving", "warm", "hot", "sunny". Furthermore suppose that the current context is in the following states: "90210", "saturday", "afternoon",



From The Ritz To The Rubble
from Whatever People Say I Am,
by Arctic Monkeys

[space]: outside, 90024
[entropy]: calm
[time]: saturday, night
[weather]: fair, warm
[kinesis]: static
[rating]: 2.5

Figure 5: Lifetrak user interface.

"walking", "calm", "warm", "cloudy". Thus the song would receive a rating of two. Since the current context can never encompass more than seven states, the maximum song rating is seven.

Figure 3 shows the XML schema associated with the context information and how rankings are represented. Figures 7 and 8 in the Appendix contain examples of actual XML files used in Lifetrak.

5.3 User Feedback

User feedback is an important aspect of Lifetrak. While listening to a song, the user may want to alter Lifetrak's tendency to play the song in the current context. Rather than force the user to explicitly edit the song database file, Lifetrak provides two user feedback options for dynamic song rating change. These options are shown in figure 4. First is the overall context equalizer which allows the user to specify the degree to which each of the context types should influence song ratings. For example, if the user temporarily decides that the entropy of his environment is more important than the weather, he or she can quickly adjust song ratings accordingly using the equalizer. Essentially the equalizer sets a weighting value between one and zero for each context type (location, time, etc.). Figure 9 in the Appendix shows an example equalizer XML file. Before Lifetrak calculates song ratings, it multiplies each state value in the current context by the equalizer weight for its context type. These modified current context values are then used to calculate song ratings as described above. Referring back to the example used in Section 5.2, suppose that the equalizer weights for time and weather are 0.5 and 0.5 respectively. Now the song will have a rating of one.

Lifetrak's second feedback mechanism provides the ability to easily adjust the context preferences for the song being played. The user can click a button to indicate that the song is appropriate, "love it", or inappropriate, "hate it", for the current context. For each state in the current context, Lifetrak either increases or decreases the value of the corresponding tag in the song's database entry. Again, returning to the example in Section 5.2, suppose the user is enjoying the song and clicks the "love it" button. As a result, the value of the tags "90210", "afternoon", "walking", "calm", "cloudy", for the song, will change from zero to 0.1. The "saturday" and "warm" tags cannot be increased any further because they already have a value of one. In effect, this function alters the likelihood that the playing song will be heard again in the current or similar contexts. Theoretically, one could forego tagging and slowly build preferences using feedback only.

6. MUSIC PLAYER

Lifetrak is a music player designed specifically for today's mobile user. While other portable music players require frequent and laborious input from the user, Lifetrak utilizes its context-awareness to minimize this interaction. The result is a cleaner, simpler interface which still offers powerful control options to the user. The following sections describe Lifetrak's graphical and hardware-key user interfaces, the details of music player operation, and a low-level overview of how Lifetrak handles music decoding.

6.1 Mobile User Interface

Lifetrak's graphical user interface is built using the GTK+ 2.6 toolkit and the Hildon UI, which provides additional widgets and themeing capabilities that match the Maemo style. The GUI is implemented in C rather than Python for several reasons. First, execution speed is a significant factor when developing applications for the Nokia 770 because the device is limited to a 220 MHz ARM processor and 64 MB RAM. Since Python is an interpreted language, it is computationally more intensive than C. The context engine and ratings generator are written in Python for its parsing capabilities, but these tasks consume most of the device's resources by themselves. Second, the application requires threading in order to update the GUI in the foreground while music is

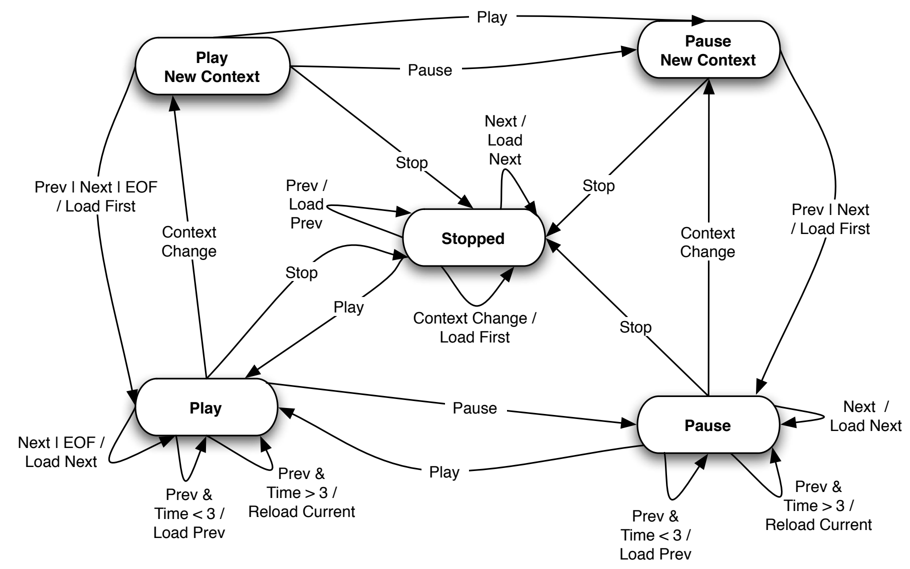


Figure 6: Lifetrak player state diagram.

played in the background. The C API of the GStreamer multimedia library handles threading efficiently, but unfortunately threading is a tricky matter for PyGTK and the python binding for GStreamer. Finally, C provides access to more extensive features in the GTK+ toolkit.

The Lifetrak GUI is laid out as follows. The Lifetrak icon and slogan are displayed in the upper left corner of the GUI. The area to the right of the icon contains the title, album, and artist of the currently queued song. The application uses the libid3tag library to retrieve this metadata from the song file's ID3 tag. While the song is playing or paused its information is displayed in dark, black text, but when stopped its information fades to gray. The large area below the song metadata contains the user's current contexts and the rating of the current song. Although the user assigns context preferences to each song when the song library is first imported, over time the user may forget these decisions. By displaying the current context and rating, Lifetrak reminds the user why a specific song is being played at a particular instance. Seeing this information also can help the user assign preferences more accurately to songs added in the future. To the left of context display is the volume slider bar, which controls the volume of the currently playing song (independent of the system volume control).

Below the context area, along the bottom of the GUI, is a toolbar containing the primary user controls. It consists of standard music player features such as a skip-to-previous button, stop button, play/pause button, skip-to-next button, and seekbar. The seekbar displays the total time, current time, and current position of the loaded song. Via the seekbar, the user can scroll to any point within the currently playing song. The toolbar also includes unique "love it" and "hate it" buttons. These offer the user a simple way to provide feedback about the current song. In an effort to make Lifetrak even more convenient for mobile users, each GUI control is mapped to a Nokia 770 hardware key. Hence one can operate Lifetrak without viewing or touching the visual display.

Noticeably absent from the Lifetrak GUI is a playlist window or file-chooser dialog box; this is intentional. These GUI features are relics of conventional portable music players. Although Lifetrak's back-end context processes do generate a new playlist each time the context changes, the user does not not need to see it. Lifetrak is designed so that the current song is likely to be enjoyed by the user in the given context. In the event that the current song is unsatisfactory, the user can simply skip ahead to the next best song. Even while the player is stopped, the user can search through the playlist by skipping ahead and behind.

Finally, the pull-down menu-tab is located in the top left corner of the GUI. Currently the menu includes a button to invoke the context equalizer dialog box, a button to invoke full screen mode, and a button to close the application. In the future, the menu will include a help section and several new user preference settings.

6.2 Player Operation

Lifetrak is designed to behave similar to a normal portable music player with several modifications to leverage its context-aware capabilities. When Lifetrak is launched from the Maemo desktop, the first song in the playlist is automatically loaded in the 'stopped' state. The user simply presses the 'play/pause' button to begin listening.

In the case that a change in context has not occurred, the primary control buttons function as they would in a standard software music player. When the player is in the 'stopped' or 'paused' state, pressing the 'play/pause' changes the state to 'playing'. When the player is in the 'playing' state, the 'play/pause' button changes the state to 'paused', and the player saves its position in the current song. When the 'stop' button is pressed in any state, the state is changed to 'stopped' and the player resets its position to the beginning of the current song. Regardless of the player's state, pressing the 'skip-to-next' button loads the next song in the playlist. If the player's current song position is less than three seconds (this includes the 'stopped' state), pressing the 'skip-to-previous' button loads the previous song in the playlist. However, if the current song position is greater than three seconds, the player resets its position to the beginning of the current song. In either case, the player retains its state. In the event that the user skips past the end of the playlist, the player wraps back to the first song in the playlist.

Player operation becomes slightly more complicated when a change in context occurs. The music player periodically checks the status of the playlist XML file at a rate of 1 Hz. If the file is modified, the player immediately updates the context information displayed in the GUI, and also displays an info-box to notify the user of the context change. If the player is in the 'stopped' state, then it immediately loads the first song in the new playlist. However, if the player is 'paused' or 'playing', it is fair to assume that the user would prefer to finish listening to the current song. In these situations the player raises a flag to signify that the context has changed. When the current song reaches end-of-file, or if the user presses 'stop', 'skip-to-previous', or 'skip-to-next' the player loads the first song in the new playlist.

6.3 Music Decoder

Due to Lifetrak's unique features, we chose to implement our software music player from scratch rather than build Lifetrak on top of a preexisting player. Although of minimal research value, this process presented somewhat of a challenge because of the lack of example multimedia applications for the Maemo development environment. Lifetrak's music playing functionality is built on top of the open-source GStreamer multimedia framework. Playing a song in GStreamer is done by assembling the source file, the desired compression decoder, and the audio sink into a pipeline. Currently Lifetrak only supports song files in the MP3 format. However we plan to support Ogg-Vorbis, WAV, AAC, and MP4 files in the future. GStreamer and Maemo can handle each of the aforementioned compression standards, so expansion should require minimal effort. The GStreamer API also provides functions for setting the volume, getting the position, and setting the position of the current song. Finally, the Nokia 770 does not have an independent sound card, but instead uses a dedicated digital signal processor (DSP) to decode and mix audio streams for the sound hardware. The DSP minimizes load on the main processor and prolongs battery life considerably.

7. DISCUSSION

There are many features that need to be added to Lifetrak in order to make it more usable and richer in terms of user experience. The following section details some of the future work plans for the Lifetrak project. Specifically, details about concepts to expand on are presented.

7.1 Context Manager

7.1.1 GUI Interface

The GUI interface for Lifetrak is important if it is to be easily operated by mobile users. There are several GUI enhancements that need to be made for Lifetrak in general. One of the fundamental interfaces that will be developed is a component for the song database context tagger. Currently, to tag contexts to songs (associate contexts for specific songs) one must edit a XML file manually. If a user's song database is very large, this manual operation will be inefficient. Thus, we imagine two ways to solve this problem. In one type of interface the user "drags and drops" context tags on songs. In addition, in order to manage tags and songs in a more cohesive fashion, an iTunes type interface can be created in which the song database is displayed indexed by characteristics such as musical attributes, context tags, and other properties related to frequency of song play. Also, searching capabilities based not only on song attributes but also on context information should be created. Finally, one can imagine only indicating the contexts in which to not play a song. This may enable users to spend less time configuring the song database.

7.1.2 Tagging Techniques

Another idea to explore is whether song tags should directly represent context states, as they currently do. A more intuitive approach is to add a layer of indirection. One would tag songs with qualities or emotions normally associated with music. As an example, tags such as "happy", "sad", or "upbeat" can be created. Then a mapping would occur to correspond these qualities to the specific contexts. This would probably make it easier for users to make the connection between songs and contexts in the configuration phase. Also, this idea has the potential to save time for users as well. It is likely that users will only create a few description or emotion tags and associate those tags to many contexts. So, initially the user must link descriptions or emotions to corresponding contexts, and after a new song is added, the user simply tags it as "upbeat", and it will be played in the appropriate situations.

7.1.3 Collaborative Tagging

In addition to giving the user additional tagging techniques, the idea of using preexisting tagging infrastructures needs to be analyzed. There are many services that already have tags on songs based on the analysis of the music itself. Furthermore, there are services that enable users to tag certain songs and then share their preferences so that songs for collective tags can be queried and listed [9]. One can imagine giving users the option of listening to similar songs based on the data that is provided by these services. Also, if there are tags already produced that are named by the contexts generated by Lifetrak, then that information can be used to figure out potential song recommendations. Overall, the idea of tagging songs and managing songs is a space that needs to be further explored.

7.2 Context Generation

Currently, the Nokia 770 requires multiple external modules to provide the necessary sensing modalities for the context engine. For example, location and speed is measured by an external GPS unit. When WiFi is not available, a Bluetooth and GPRS enabled mobile phone is needed to access RSS feeds on the internet. A single modular hardware component that could provide these modalities would make Lifetrak more convenient for mobile users.

Furthermore, much context information is not fine-tuned to the specific state of the user. Lifetrak could provide a more personalized user experience if the context generation occurred locally. Some of these services could be provided by leveraging GPS coordinates as opposed to a zip code transformation. For instance, getting traffic or weather information from the Internet resources using the GPS location values would make it a bit more specialized for the user. Also, one could also make the argument that the sensing could be local. There are several sensors that can be used to get better context locally, such as photodiode or humidity sensors, to get more granular information. This information can be used solely if connectivity is not available or could augment high level context information.

7.3 User Feedback

To influence what songs should be played in a certain context once a playlist has been generated by the context engine, one must use the "love it" and "hate it" buttons or changing values of the context equalizer. But what songs are actually picked makes a big difference on how much a player gets used. Thus, giving the user the ability to adjust the playback in a certain context is important. One can imagine two contexts being very similar and thus the same set of songs get played in both contexts. The idea of adding freshness and jitter factors would enable the user to mixup what songs are played back in a controlled fashion while still following the fundamental idea of context-aware music generation. Basically, the freshness factor enables a user to prevent recently played songs from being re-played in high frequency even if their rating is high when contexts change. The jitter factor adds a normalized change to the ratings in a certain context's playlist so that playback of songs is not in the same chronological order. Both these factors should be incorporated in the player GUI as slider bars that users can change.

7.4 Other Music Sources

Once context information is obtained, one does not have to be limited to the song database of the user for a music source. Instead, the idea of using the context information to rank other forms of media including sources that are available in a streaming form via the Internet or even radio stations (satellite and normal) is a logical next step. The other forms of media have to be annotated by the user in some form to correspond to a context, but once this process is done they can easily be used as sources for music.

8. CONCLUSION

One of the most popular forms of entertainment for mobile users is listening to music. Currently the user's music listening experience is dictated by predefined playlists, random selection of songs, or tempo matching solutions. For the most part, listening enjoyment requires explicit user input. Lifetrak uses technologies related to sensing and applies it to a music player so that the user can spend less time choosing what music to play and more time enjoying it. Music is tagged by the user to be played in a certain context. The context is obtained by using various resources, and music is correlated with the user defined tagging. The context information includes the user's location, time, speed of movement, entropy of the environment, and weather. Since Lifetrak is designed with the mobile user in mind, it provides a simple, easy to use GUI that can be manipulated by users on the go. Lifetrak also has several shortcuts that enable playback control without explicit user attention. Overall, Lifetrak provides a user-centric music listening experience by using context to influence what songs should be played in a mobile setting.

9. REFERENCES

[1] K. Blanchette. Effects of MP3 Technology on the Music Industry: An Examination of Market Structure and Apple iTunes. *College of the Holy Cross. Retrieved September*, 30:2004, 2004.

[2] Apple. Apple ipod family. http://www.apple.com/ipod, 2006.

[3] S.I. Tamminen, A.I. Oulasvirta, K.I. Toiskallio, and A.I. Kankainen. Understanding mobile contexts. *Personal and Ubiquitous Computing*, 8(2):135–143, 2004.

[4] Sony. Sony walkman bean mp3 player. http://www.sonystyle.com/walkman, 2006.

[5] G.T. Elliott and B. Tomlinson. Personalsoundtrack: Context-aware playlists that adapt to user pace. In *Conference on Human Factors in Computing Systems*, pages 298–304. CHI, April 2006.

[6] G. Wijnalda, S. Pauws, F. Vignoli, and H. Stuckenschmidt. A personalized music system for motivation in sport performance. *Pervasive Computing, IEEE*, 4(3):26–32, 2005.

[7] P. Riddle. Tags: What are They Good For? http://www.ischool.utexas..edu, 2005.

[8] J. Crossett. Social Classification of Data. http://jamescorssett.com, 2006.

[9] LastFM. Last.fm, the social music revolution. http://www.last.fm, 2006.

[10] R. Want, B.N. Schilit, N.I. Adams, R. Gold, K. Petersen, D. Goldberg, J.R. Ellis, and M. Weiser. An overview of the PARCTAB ubiquitous computing experiment. *IEEE Personal Communications*, 2(6):28–43, 1995.

[11] G.D.B. Abowd, C.G.B. Atkeson, J.B. Hong, S.B. Long, R.B. Kooper, and M.B. Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, 1997.

[12] M.S. Pandit and S. Kalbag. The selection recognition agent: instant access to relevant information and operations. *Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 47–52, 1997.

[13] A.K. Dey. Context-aware computing: The CyberDesk project. *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, 1998.

[14] P. Dourish. Seeking a Foundation for Context-Aware Computing. *Human-Computer Interaction*, 16(2, 3 & 4):229–241, 2001.

[15] JA Landay and TR Kaufmann. User interface issues in mobile computing. *Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on*, pages 40–47, 1993.

[16] S. Sarker and J.D. Wells. Understanding mobile handheld device use and adoption. *Communications of the ACM*, 46(12):35–40, 2003.

[17] Nokia. Nokia 770. http://www.nokia.com/770, 2006.

APPENDIX

A. XML FILE EXAMPLES

Figures 7, 8, and 9 are XML files used in Lifetrak.

```
<lifetrack>
    <music>
        <song>
            <id>1</id>
            <file\_name>file1</file\_name>
            <artist\_name>artist1</artist\_name>
            <song\_name>song1</song\_name>
            <album\_name>album1</album\_name>
            <ranking>0.000000</ranking>
            <context>
                <space>
                    <info name=”90024” value=”0.9”>
                    </info>
                    <info name=”90210” value=”0.0”>
                    </info>
                    <info name=”inside” value=”0.3”>
                    </info>
                    <info name=”outside” value=”0.0”>
                    </info>
                </space>
                <time>
                    <info name=”morning” value=”0.9”>
                    </info>
                    <info name=”afternoon” value=”1.0”>
                    </info>
                    <info name=”evening” value=”0.0”>
                    </info>
                    <info name=”night” value=”0.0”>
                    </info>
                    <info name=”monday” value=”0.9”>
                    </info>
                    <info name=”tuesday” value=”1.0”>
                    </info>
                    <info name=”wednesday” value=”1.0”>
                    </info>
                    <info name=”thursday” value=”0.0”>
                    </info>
                    <info name=”friday” value=”0.9”>
                    </info>
                    <info name=”saturday” value=”0.0”>
                    </info>
                    <info name=”sunday” value=”0.9”>
                    </info>
                </time>
                <kinetic>
                    <info name=”static” value=”0.9”>
                    </info>
                    <info name=”walk” value=”0.0”>
                    </info>
                    <info name=”run” value=”1.0”>
                    </info>
                    <info name=”driving” value=”0.9”>
                    </info>
                </kinetic>
                <entropic>
                    <info name=”calm” value=”1.0”>
                    </info>
                    <info name=”moderate” value=”0.0”>
                    </info>
                    <info name=”chaotic” value=”0.9”>
                    </info>
                </entropic>
                <meteorological>
                    <info name=”rain” value=”1.0”>
                    </info>
                    <info name=”snow” value=”1.0”>
                    </info>
                    <info name=”haze” value=”1.0”>
                    </info>
                    <info name=”cloudy” value=”0.9”>
                    </info>
                    <info name=”sunny” value=”0.9”>
                    </info>
                    <info name=”clear” value=”1.0”>
                    </info>
                    <info name=”frigid” value=”0.0”>
                    </info>
                    <info name=”cold” value=”1.0”>
                    </info>
                    <info name=”moderate” value=”0.0”>
                    </info>
                    <info name=”warm” value=”0.3”>
                    </info>
                    <info name=”hot” value=”1.0”>
                    </info>
                </meteorological>
            </context>
        </song>
    </music>
</lifetrack>
```

Figure 7: Lifetrak song database file containing one song.

```
<?xml version=”1.0” ?>
<lifetrack>
    <playlist>
            <music>
                <song>
                    <id>8</id>
                    <file\_name>file8.mp3</file\_name>
                    <ranking>1.5</ranking>
                </song>
                <song>
                    <id>9</id>
                    <file\_name>file9.mp3</file\_name>
                    <ranking>1.4</ranking>
                </song>
                <song>
                    <id>10</id>
                    <file\_name>file10.mp3</file\_name>
                    <ranking>1.1</ranking>
                </song>
                <song>
                    <id>7</id>
                    <file\_name>file7.mp3</file\_name>
                    <ranking>1.1</ranking>
                </song>
                <song>
                    <id>6</id>
                    <file\_name>file6.mp3</file\_name>
                    <ranking>1.1</ranking>
                </song>
                <song>
                    <id>5</id>
                    <file\_name>file5.mp3</file\_name>
                    <ranking>1.1</ranking>
                </song>
                <song>
                    <id>4</id>
                    <file\_name>file4.mp3</file\_name>
                    <ranking>1.1</ranking>
                </song>
                <song>
                    <id>3</id>
                    <file\_name>file3.mp3</file\_name>
                    <ranking>1.1</ranking>
                </song>
                <song>
                    <id>1</id>
                    <file\_name>file1.mp3</file\_name>
                    <ranking>0.96</ranking>
                </song>
                <song>
                    <id>2</id>
                    <file\_name>file2.mp3</file\_name>
                    <ranking>0.7</ranking>
                </song>
            </music>
        <context>
            <space>
                <info name=”inside” value=”1”/>
                <info name=”90024” value=”1”/>
            </space>
            <time>
                <info name=”friday” value=”1”/>
                <info name=”night” value=”1”/>
            </time>
            <kinetic/>
            <entropic/>
            <meteorological>
                <info name=”fair” value=”1”/>
                <info name=”warm” value=”1”/>
            </meteorological>
        </context>
    </playlist>
</lifetrack>
```

Figure 8: Lifetrak playlist XML file.

```
<lifetrack>
    <equalizer>
        <info name=”space” value=”0.3”>
        </info>
        <info name=”time” value=”0.4”>
        </info>
        <info name=”kinetic” value=”0.5”>
        </info>
        <info name=”entropic” value=”0.6”>
        </info>
        <info name=”meteorological” value=”0.8”>
        </info>
    </equalizer>
</lifetrack>
```

Figure 9: Lifetrak equalizer file.