ERELT: A Faster Alternative to the List-Based Interfaces for Tree Exploration and Searching in Mobile Devices

Abhishek P. Chhetri
Computer Engineering Program
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX 65080-3021, USA
achhetri@utdallas.edu

Kang Zhang
School of Software Engineering
Tianjin University, Tianjin, China
Department of Computer Science
University of Texas at Dallas
Richardson, TX 65080-3021, USA
kzhang@utdallas.edu

Eakta Jain
Texas Instruments
Dallas TX, USA,
Department of Computer Science
University of Texas at Dallas
Richardson, TX 65080-3021, USA
eakta.jain@gmail.com

ABSTRACT
This paper presents ERELT (Enhanced Radial Edgeless Tree), a tree visualization approach on modern mobile devices. ERELT is designed to offer a clear visualization of any tree structure with intuitive interaction. We are interested in both the observation and navigation of such structures. Such visualization can assist users in interacting with a hierarchical structure such as a media collection, file system, etc.

In the ERELT visualization, a subset of the tree is displayed at a time. The displayed tree size depends on the maximum number of tree elements that can be put on the screen while maintaining clarity. Users can quickly navigate to the hidden parts of the tree through touch-based gestures. We conducted a user study to evaluate this visualization for a music collection. Test results show that this approach reduces the time and effort in navigating tree structures for exploration and search tasks.

Categories and Subject Descriptors
H.5.2 [**User Interfaces**]: Graphical user interface (GUI), H.1.2 [**User/Machine Systems**]: Human factors, I.3.6 [**Methodology and Techniques**]: Interaction Techniques

General Terms
Algorithms, Design, Human Factors

Keywords
Hierarchy Visualization, Mobile Devices, Navigation, RELT (Radial Edgeless Tree), User Interface

1. INTRODUCTION
Current market trends show strong rise in smartphones and tablets. Smartphones can now match the processing capabilities of laptop/PCs from a few years ago with a fraction of the power usage. With emails, contacts, documents, pictures and music all stored in the cloud, one no longer needs to sit in front of a personal computer to access data. There are, however, still many challenges in mobile computing, such as smaller screens and lack of separate input devices such as keyboard.

VINCI '13, August 17 - 18 2013, Tianjin, China

http://dx.doi.org/10.1145/2493102.2493109

Although the screen resolution in mobile devices has been increasing, in terms of screen size they are still much smaller than laptops/PC monitors. This makes it difficult to present tabular and hierarchical structures in mobile device when a large proportion of application data are hierarchical in nature. For example, a file system is a hierarchical structure, and a file list within a folder is usually displayed in tabular format. A multimedia collection such as music, pictures, videos, etc. may exist in hierarchical structures, and is usually displayed in tabular form in laptops/PCs.

Apart from the presentation issues, how we interact with these data structures in mobile devices is also challenging. Modern mobile devices are mostly equipped with touch screens, and soft keyboards. Any keyboards or buttons displayed on screen take space, which is already at a premium. Thus, it is necessary to come up with intuitive methods of interaction without sacrificing screen area for input. Most hierarchical structures are represented by lists in mobile devices. Lists offer fast interaction but can only display single level under one node at a time. This paper presents a technique for visualizing and navigating hierarchical structures on mobile devices that focuses on the two issues presented above.

1. Maximal utilization of screen area to display hierarchical structures.

2. Intuitive interaction mechanism that allows rapid navigation and exploration of the structures.

The research aims to utilize the screen estate to display maximum possible information, without sacrificing clarity. The objective is to use touch technology in most modern smartphones to implement gesture based commands that are intuitive and mimic real-world object interactions. The paper presents further enhancement over our earlier prototype [1], and a user study using media player application utilizing ERELT. Our user study shows that ERELT supports faster exploration of tree structures than traditional list based interfaces.

To date, considerable research has been done in the areas of information visualization and human-computer interaction (HCI). While many of these researches have produced a variety of visualization techniques for hierarchical structures, little previous work has focused on utilizing visualization as UI elements. Our contribution is a hierarchy visualization technique for small screens with a practical approach for user interaction. We extensively evaluate the ease of user interaction with the proposed ERELT visualization through a user study. Our results show that it takes significantly less time and fewer number of

touches to perform exploration and search tasks. In some cases, the number of touches is reduced by nearly 50%. This suggests that ERELT is a more appropriate interface for users for interacting with hierarchical data than, the traditional list interface.

The next section reviews the previous research on the subject. We discuss previous works on which our research is based including our own, explore their shortcomings and propose improvements. Section 3 describes our solution. Section 4 presents algorithms and implementation details, followed by the evaluation of the research through user tests. Finally, Section 5 presents conclusion and ideas for future work.

2. RELATED WORK

Hierarchy visualization techniques can broadly be classified into two types - implicit and explicit [2]. Explicit visualization techniques display the edges between the connected vertices of the hierarchy. In contrast, implicit techniques, also known as enclosure techniques, show hierarchical relations through shape, location and area of vertices [3]. The implicit techniques are usually shape filling and better cover the display area than explicit techniques [2].

One of the most commonly used implicit visualization techniques is the Tree Map [4][5]. A Tree Map maximizes area utilization by enclosing child nodes in their parent node. It is good at displaying many leaf nodes in a small space but cannot clearly show parent-child relationships. Thus, it is difficult to navigate between different levels of a tree. Data Jewelry-Box [6] is a slight variation on traditional Tree Maps. It attempts to better represent intermediate nodes at the cost of maximum area usage. The circular partition scheme [7] is another area division algorithm where the thickness of division lines represents levels of the nodes in the tree hierarchy.

The intermediate node relations are more effectively displayed in implicit layout created through adjacency instead of enclosure [3]. Sunburst [8] and Aggregate Tree Map [9] are examples of this type of layout. Here, child nodes are drawn adjacent to one or more sides of their parent node. In such a layout, the deeper the tree level, more area it gets. This helps because trees usually have more nodes at deeper levels. However, both Sunburst and Aggregate Tree Map are independent of the shape of the display area; thus, do not fully utilize the screen space. For example, a Sunburst layout drawn at the center of a rectangular screen of a smartphone leaves large spaces at the two ends while fully occupying shorter width of the screen. Also, some of these and other radial division algorithms, such as pitTree [10], are designed for data visualization without many provisions for user interaction for exploration.

Radial Edgeless Tree (RELT) [11] [12] [13] is another adjacency-implicit layout technique. It combines the best features of Tree Map and Sunburst. Like Tree Map, RELT divides the given display area into several vertices to maximize the area utilization, and like Sunburst, it allocates area for intermediate nodes.

However, our tests reveal some particular cases where RELT did not produce good visualization output. For example, if the second level of the tree has only one or two child nodes then the area of the root node would totally collapse or be very small. Since the shape of any node depends on radial lines dividing area between its children, RELT sometimes cannot produce proper-shaped nodes. Shape inconsistency in RELT forces the node labels to fall in different directions hindering readability.

We previously built and explored a prototype ERELT (Enhanced RELT) system [1] based on RELT. In ERELT the node shape depends on a set of proportionate rectangles and radial division lines, to ascertain that the node shapes are always in a proper form. This allows the node labels to be written in a consistent manner yielding high readability.

Beside visualization layout choices, user interactions and implementation issues should also be considered. When designing InterRing [14], authors had attempted to establish a set of functionality that a good hierarchy visualization and navigation system should provide. For example, users should be able to select any particular node and interact with it. Different nodes or levels should disappear or reappear based on use to preserve memory and screen area. There should be a mechanism for users to access hidden nodes by panning or rotating the tree view. Karstens, *et al.* [15] proposed another set of considerations for tree visualization aiming at effective implementation. They suggest limiting the number of tree elements drawn on the screen at a time. Visualization in 2D graphics is much faster than in 3D. Also the paper encourages the use of simple graphical primitives like straight lines instead of curves.

Additional design elements such as animation is also known to drastically change the user experience [16]. Correct use of animation can make GUI more intuitive and the delay in application response appears shorter.

Based on our experience and user feedback from the initial prototype, we made several improvements to RELT. These changes affected both the performance and presentation. We added color effects to indicate different node groups and to provide context when users scroll through various nodes in the tree. Visual effects were added to make tree elements appear like interactive buttons rather than plain drawings on the screen. To account for the performance cost of enhanced graphics, the visualization algorithm was completely re-designed to make it much faster. These changes are discussed in the next section.

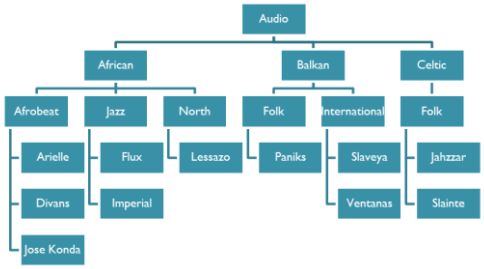

Figure 1: Sample Music Library Segment (Language, Genre & Artist Levels)

3. DESIGN

The main idea behind ERELT is to display a hierarchical structure on a small screen where users can view multiple levels of hierarchy in a single display and interact with it. This technique divides the screen proportionally into various levels. These divisions form multiple levels of rectangles one inside another. The innermost rectangle contains the root. The area between the innermost rectangle and the next rectangle contains the level under root, and so on. This implies that the deeper levels of the tree, which have more number of nodes, are represented by larger rectangles. Radial lines starting at root separate the nodes within a level. The space between two adjacent radial lines gets wider farther from the root, which is ideal because there are

usually more nodes at levels farther away from the root. Figure 1 shows a sample music library and Figure 2 shows its ERELT representation.

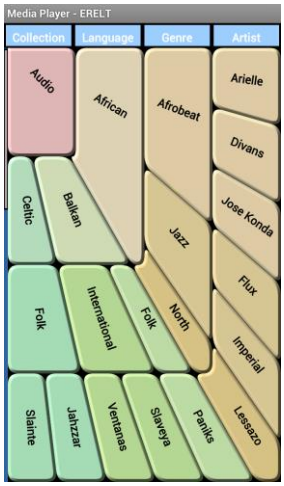

Figure 2: ERELT Visualization of the Structure in Figure 1

Unlike our previous prototype [1], the root has been fixed at the top-left corner of the screen. Although it is more intuitive to place the root at the center of the layout, our tests showed that placing root at the center made the central region more crowded because most smartphones have much larger length than width as drawn in ERELT prototype [1] in Figure 3.

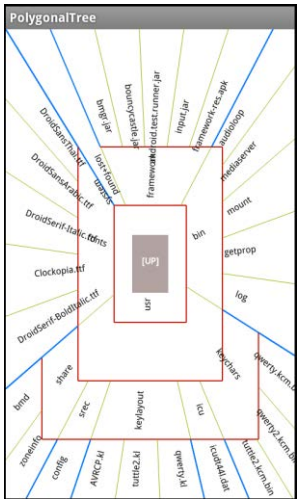

Figure 3: Center-rooted Layout in An Earlier Prototype: Less Space for Labeling in Mid-Area

Having the root at the center and nodes populating at all sides of the root creates wider and shorter nodes. Our labeling scheme, however, works well with narrower and longer nodes. Though we adopt corner-rooted layout for this user study, we can easily generate center-rooted layout, which is more intuitive and better suited for wider screens, e.g. tablets and PCs.

3.1 Layout

The display area is divided into various polygons, each representing a node in the hierarchy. Two important considerations are made:

- The division should consider the size and the shape of the display area to maximize the space utilization.
- The division should distribute the area such that each node is given a minimum space to fill the label.

Before starting the division algorithm, a setup process is performed:

```
void setWeight (Node node) {
   if (node is Leaf_Node) {
      node.weight = 1;
   } else {
      for each (Node n in node.childNodes){
         setWeight (n);
         node.weight += n.weight;
} } }
```

Setting the leaf nodes weight based on any other property will result in a display tree with area distribution that reflects that property.

We start by selecting the top-left corner as the center for the visualization tree. This choice was arbitrarily made for the current user study and is not essential for the algorithm to work. The algorithm works for the root at any location in the display area.

We also start with H virtual rectangles, where H is the total height of the tree measured by number of levels. For large trees, H can be limited to a maximum threshold value. Handling large trees will be discussed in details later. These rectangles are "virtual" because they are not yet drawn onto the screen but only exist as variables in the algorithm. This definition of “virtual” will apply to any lines or rectangles. The H virtual rectangles are created in the following steps:

1. Connect four corners of the display area to the visualization center by four virtual lines. These four lines will be referred to as “sector lines”.
2. Divide the four lines into H equal segments (one for each level).
3. Form virtual rectangles by connecting four division points at each level.

Each virtual rectangle has the same aspect ratio as that of the display area. It represents one level of the tree, with the size proportional to the level of the tree. The four sections of the display area divided in Step 1 by the sector lines will be referred to as *sectors*. Virtual rectangles for corner-rooted layout for a tree of 4 levels are shown in Figure 4. This layout helps utilize the entire screen space.

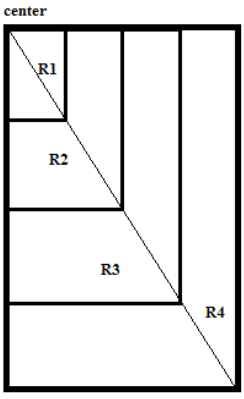

Figure 4: Virtual Rectangles for Corner-Rooted Layout for A Tree of Four Levels

For any given node, the division algorithm listed below calculates area and draws each node as a polygonal button. Drawing the nodes onto the screen is handled by an algorithm where the node area is divided into smaller sections by radial lines to represent child nodes. This algorithm runs recursively for each node and its child nodes until the entire tree is plotted. The following symbols are used in the algorithm:

N = given node

$\mathbf{N_{parent}}$ = parent of the given node

$\mathbf{Rect_L}$ = virtual rectangle at level **L**

$\mathbf{Rect_{L+1}}$ = virtual rectangle at level **L+1**

$\mathbf{Line_{start}}$ = radial line forming the starting boundary of **N**

$\mathbf{Line_{end}}$ = radial line forming the ending boundary of **N**

$\mathbf{W_N}$ = weight of **N**

$\mathbf{W_{parent}}$ = weight of $\mathbf{N_{parent}}$

$\mathbf{W_{cumulativeN}}$ = cumulative weight of all the sibling nodes processed before **N**, under $\mathbf{N_{parent}}$

The following function is called to recursively draw every node:

```
void drawNode (Node N) {
  if (N is Root_Node){
    // Innermost rectangle as the root
    plotNodeOnAreaEnclosedByShapes (Rect1);
  } else {
    Get WN, Wparent, WcumulativeN;
    // Divide the area under Nparent radially into Wparent parts with
    // equal areas, and get dividing lines
    Lines[] = divideArea (Nparent, Wparent);
    // Select the starting and ending radial lines for current node
    // based on cumulative weight of previous siblings and its own
    // weight
    Linestart = Lines[WcumulativeN];
    Lineend = Lines[WcumulativeN + WN];
    // Draw the polygonal node as the area between RectL, RectL+1,
    // Linestart & Lineend
    plotNodeOnAreaEnclosedByShapes (RectL, RectL+1, Linestart,
Lineend);
  }
}
```

3.2 Scalability

In theory the layout algorithm works for a tree of any size. However, for applications that require labels on all the nodes, drawing a large number of nodes in a single screen makes labels unreadable and the visualization loses clarity. To handle such situations the algorithm enforces some limitations.

1. Level threshold is the total number of levels displayed on the screen at a time. For a normal sized smartphone screen (between four and five inches) we found four to be the proper level threshold. For a smartphone screen less than four inches the threshold value of three is better for readability. Larger devices (5-6 inches) can properly display trees with level threshold of five or six.

2. Branching threshold is the total number of child nodes displayed under each parent at a time. The threshold value of three was found to be good for general cases. For smaller trees however, the value of four works just as well.

After the two thresholds are set, the tree structure is pruned using these values and only the sub-tree is included each time it is displayed. The remaining tree is considered hidden. To display any part of the tree, a node in that part is selected as the root and the corresponding sub-tree is displayed.

Wherever the branching threshold is applied, arrow indicators are used to show sibling nodes hidden to the left or right of the displayed nodes, as shown in Figure 5.

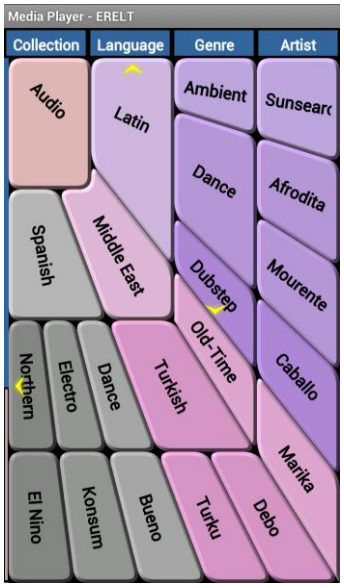

Figure 5: The Arrow Markers Indicating Direction of Nodes Hidden due to Branching Threshold

3.3 Coloring

Since the number of nodes increases away from the root, it becomes difficult to distinguish two nodes of different parents and two nodes of the same parent at deeper levels. To avoid confusion and provide a clear structure at the first glance, the algorithm uses a dynamic coloring scheme that allocates different colors to the nodes based on their "nearness" to each other. Here, nearness between two nodes at the same level refers to how far the tree needs to be traversed in the direction of the root before finding a common ancestor node for the two nodes. If there are three nodes side by side with two of the nodes sharing the same parent, then the two sibling nodes have less color variation than the third node.

We use the HSL color space to distribute colors to the nodes. HSL has three components - Hue (color), Saturation (Intensity of color), and Lightness (Amount of black or white in the color). The algorithm assigns the nodes with the same color as their parent node and then changes hue, saturation or lightness based on level and position.

Coloring Algorithm:

```
setColor (Node n) {
  if (n is Root_Node) {
    // set a predefined color for root
    n.color = HSL (h, s, l);
  else if (n.level == Leaf_Node)
    n.color = n.parent.color;
  else if (n.level == Level_2)
    n.color = HSL (hparent+N*Δh, sparent, lparent);
  else if (n.level == Level_3)
```

```
    n.color = HSL (h_parent, s_parent+N*Δs, l_parent);
  else if (n.level == Level_4)
    n.color = HSL (h_parent, s_parent, l_parent+N*Δl);
}
```

In the coloring algorithm, changing colors of leaf nodes of the same parent is unnecessary. A leaf node inherits its parent node's color so it will always be different from the leaf nodes under other nodes. The algorithm is designed to display for up to five levels. As discussed above, four levels are appropriate for displaying readable labels.

3.4 Labeling

The ERELT layout is designed with narrow and long nodes. This allows a node to be labeled along the length of the node. The consistent label style makes it easy for users to scan through the tree. For each node the line bisecting the two radial lines forming the node is calculated. The label is placed along the bisecting radial line between the two neighboring rectangles forming the node.

3.5 Navigation

The algorithm handles large trees by only displaying a sub-tree to maintain clarity and readability. Thus it is essential to have an effective interaction mechanism that allows users to navigate the structure and display the desired nodes. The design supports touch-screen interactions available on most smartphones, with intuitive gestures making the response of the visualization and interactions instantaneous.

The navigational gestures/actions from users are interpreted as *Tap* or *Scroll* gestures. Tap actions allow users to navigate to the tree levels hidden in the current view due to the level threshold. Scrolling gestures allow users to navigate to the sibling nodes hidden in the current view due to the branching threshold.

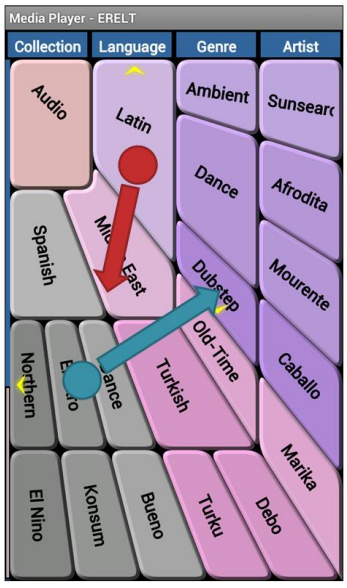

Figure 6: Left/Right Scrolling

3.5.1 Tap

There are three types of tapping interactions:

1. Whenever users tap on a node in the tree, that node becomes the new root of the tree and a new layout will be drawn.

2. Tapping within the root node triggers an "Up" command. If the current root is not the root of the main tree then tapping on it sets the parent of the current root as the new root and the new tree layout is drawn.

3. Tapping on the back button of the device triggers a "Back" command. ERELT maintains a limited history of user interactions. The back command loads the previous segment of tree into the display. This allows users to quickly move into and out of several nodes, multiple levels down the tree.

3.5.2 Scroll (Drag)

Scrolling gestures illustrated in Figure 6 occur when users touch the screen with one finger and drag the finger across the radial lines to a different point in the display, before raising the finger from the screen. This gesture starts at one of the nodes whose sibling nodes have been hidden due to the branching threshold. The direction of the drag then moves to the opposite direction of the hidden nodes to simulate the action of dragging the nodes to give room to display the hidden nodes. This gesture results in left/right scrolling of the nodes under the initial point of touch.

3.6 Complexity Analysis

Computation time for ERELT does not depend on the size of the tree because the algorithm only displays a sub-tree at a time. Thus, the level threshold and the branching threshold values determine the time complexity.

For a tree with N nodes,
  Level threshold = **d**
  Branching threshold = **b**
  Maximum nodes to be drawn = $\mathbf{b^d-1}$
Thus,
  Computational complexity = $\mathbf{O(b^d)}$
For a given device, **b** and **d** do not depend on the size of the tree and can be set as constants, therefore,
  Computational complexity = **O(1)**

3.6.1 Comparison between ERELT and RELT

Since RELT [11] [12] [13] only describes method to draw the complete tree, following assumptions are made for comparison:

1. Level threshold is larger than the tree height
2. Branching threshold is larger than the node with the largest number of child nodes.

The RELT algorithm operates by keeping one radial line around a node fixed and moving the other radial line along the display area border, one pixel at a time. At each step, the area within the two radial lines is calculated. The process continues until the area obtained is equal to the required area for the node. The number of calculations to obtain the area for a node depends on number of pixels traversed along the border. Thus, even when the tree size remains constant, the computation cost increases with the increase in the size of the display.

The ERELT algorithm computes the display area by calculating the required angle at the center between the two radial lines. Given the required area value, this angle can be directly calculated using trigonometric functions. Thus, the computation does not depend on the display size, but only on the number of nodes to be displayed.

For a clear and readable layout the number of nodes to be displayed at any level must be smaller than the pixels along the border. Thus the computation in ERELT is much cheaper than in RELT for displaying trees of the same size.

3.7 Implementation

ERELT was implemented and tested on an Android platform. It has been tested in Froyo (2.2), Gingerbread (2.3), Ice-Cream Sandwich (4.0) and Jelly-Bean (4.1) versions of Android OS. The application was written in Java to run in Dalvik VM in Android. We used Eclipse IDE with Android SDK for development. It has been tested to run in Android simulator (Android 2.2, 2.3.3, 4.0 and 4.1), Motorola DroidX (Android 2.3.3) and Samsung GS3 (Android 4.1) during the development.

For prototyping and developer testing, ERELT was built into two different Android applications: A File-system Explorer and a Media Player. These test applications also had traditional list based layout for comparison with the tree layout. An ERELT application uses Android 2D graphics functions to draw tree layout and animations onto a bitmap object. This image is then displayed on an Image View controller and presented to the user.

4. EVALUATION

4.1 User Study

In order to evaluate whether ERELT provides an efficient interface for navigation, a user study was performed to compare it against a traditional list based interface for accessing hierarchical structures. The objective of the test was to compare ERELT and List view in terms of effort and time taken to perform a set of tasks.

In the usability study of an interface, the physical effort can be estimated by the number of physical interactions required [17]. Thus, the number of touches required to perform any task was selected as the metric for the measurement of effort [18]. In addition, mental or cognitive effort also affects the user experience [17]. However, the measurement of cognitive effort requires complex supervised techniques, such as eye-tracking, which is beyond the scope of this study.

4.2 Setup

A media player application was chosen for user testing, where the media library has a hierarchical structure, composed of various levels such as language, genre, artist, albums and tracks. We chose this application as a test case because 63% of smartphone users use mobile phones as a music player [19]. This was the fifth highest use after GPS/navigation, social media, local search and news reading.

An Android application was created that allows users to navigate the music library in ERELT interface to find the track they want to listen to. Figure 8 shows the media control interface of the application. Pressing the "Add Song" button in the control interface opens up the music library in ERELT layout. The level threshold of four and branching threshold of three were chosen. "Up" and "Back" gestures were disabled in the user study to reduce the complexity of the learning curve for new users. Users can click on the root node for "Up" functionality, and use Android's "Back" button for "Back" functionality.

The application also includes an option for navigating the library through a list-based interface for comparison. The application stores logs of user interaction events with timestamps. This log can be manually or automatically uploaded to a data collection server. Each user enters an ID into the application, which helps to organize the log files in the data server.

For the study to be meaningful, all the users perform the same test. For this purpose, instead of making the application pull up media files from the user's individual smartphone, the application was created with built-in music library of 8 languages, 22 genre, 38 artists, 49 albums and 138 tracks. Including the complete media files with this library structure would have made the application huge in size. Thus, we only included short (1 minute) snippets of about 24 tracks, and associated all of 138 tracks in the library listing with one of these 24 tracks.

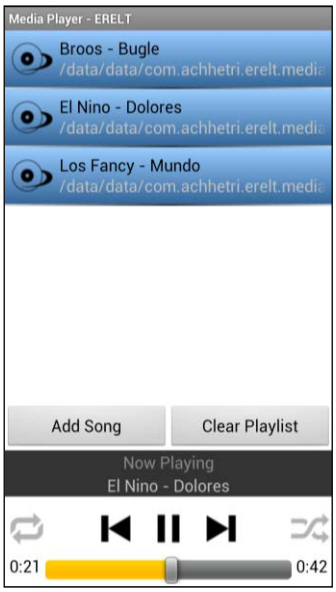

Figure 8: Media Control Interface of the Media Player Application

The music library was based on a collection of music pieces from the real international artists providing their music freely over the Internet. The advantage of using independent international artists from various parts of the world was that there was a very little chance that users were already familiar with the entire library before the test. This initial library was then altered to change the structure and names of various elements so that it would be better suited for creating a variety of tests. For example, some of branches were removed from one place and added to another to create sections of the tree that were much denser or sparser than the rest of the tree. Another reason for this renaming and restructuring was to create multiple segments of the tree that had similar structures but differently named elements. This allowed us to create comparable tests for List view and Tree view (using ERELT) that looked different on names but similar on the structure.

4.3 Test Design

The user task consisted of a paper based test and application test, consisting of four components:

4.3.1 Test for Basic Understanding:

The first part of the test provided users (called *subjects* from here on) with a simple ERELT diagram containing four levels and asked subjects to answer five basic questions about it. The purpose here was to know whether or not they had basic understanding of the ERELT layout before starting on application test. If a subject could not complete this part correctly, it meant he/she did not understand the ERELT structure, and the test data needed to be ignored.

4.3.2 Familiarization with the Music Library:

The second part of the test asked subjects to answer five questions about the music library. This would require subjects to browse the music library in the test application and perform tasks, such as counting albums with certain names, albums under certain artists, languages with certain genre, etc. This part of test was not timed and only analyzed for general correctness. The purpose of this test was to familiarize the subjects with the music library before performing the timed-tasks. We felt this step was necessary because the subjects are somewhat familiar with of most of the structures in their smartphones.

4.3.3 Timed Tasks:

The third part of the test asked subjects to find and add various tracks to the media player playlist. They had six tasks to perform in this section. In Tasks 1 & 2, the entire path from the root to the track was given. Tasks 3, 4 and 5 involved searching various parts of the tree for a certain genre or album and adding multiple tracks to the playlist. Finally, Task 6 involved adding multiple tracks from given path in the library. The time and the number of touches required to complete these tasks were stored in the log files. A list of timed tasks is shown below.

Timed Tasks Sets:

Task Set A

1. Add the track "**Etelemelo**" from the album "**Nzambe**" by the artist "**Jose Konda**" in the "**Afrobeat**" genre of the "**African**" music.
2. Add the track "**Paga**" from the album "**Elegante**" by the artist "**Konsum**" in the "**Electro**" genre of the "**Spanish**" music.
3. Add any two tracks from the "**Electro**" genre (can be from the same artist or album).
4. Add any two tracks from the "**Northern**" genre of any two different languages.
5. Add three tracks from three different "**Singles**" albums.
6. Add all tracks from the "**Rock**" genre of the "**Spanish**" language.

Task Set B

1. Add the track "**Digitale**" from the album "**Pas de tigre**" by the artist "**Divans**" in the "**Afrobeat**" genre of the "**African**" music.
2. Add the track "**Radium**" from the album "**Elegante**" by the artist "**Konsum**" in the "**Electro**" genre of the "**Spanish**" music.
3. Add any two tracks from the "**Dance**" genre (can be from the same artist or album).
4. Add any two tracks from the "**Folk**" genre of any two different languages.
5. Add three tracks from three different "**Live**" albums.
6. Add all tracks from the "**Dance**" genre of the "**Latin**" language.

4.3.4 Feedback:

The final part of the test was a feedback survey. The feedback section asked subjects their opinion on quickness of using ERELT vs. traditional list layout. In addition, we also collected their input on UI mechanism, missing features, change/enhancement requests, etc.

The aforementioned second [4.3.2] and the third [4.3.3] parts of the tests had to be done in application in both list-based layout and tree-based layout. All 11 tasks (5 from the second part [4.3.2] and 6 from the third part [4.3.3]) were designed in pairs, one for list and one for tree. These were designed in such a way that they looked different based on names of elements in the tasks but involved navigating similar structures in the tree. This variation allowed the same subject to take two tests without learning effects. To remove bias, half of the subjects were asked to perform the list view test first followed by the tree view test, and the other half had to perform the tree view test first followed by the list view test.

4.4 User Profile

Most subjects participated in the study were volunteers from a population of undergraduate and graduate students in the Computer Science, and the Arts and Technology programs at the University of Texas at Dallas, USA and School of Software Engineering at Tianjin University in China, between the ages of 18-30. About three-fourth of the subjects taking the test were completely new to the ERELT interface. The remaining quarter were familiar with the visualization technique but had not used it. Almost all the subjects used their own Android smartphones for the test, thus, were assumed to be familiar with the Android OS and the device.

4.5 Results

A total of 38 subjects participated in the test as evaluation, out of which 26 results were selected. 12 subjects results were rejected for one or more of the following reasons: incomplete test, incorrect actions, and distracted during test. A subject was considered distracted if he/she took long pauses when performing the tasks. This was measured by checking time spent in the middle of a task without any action. The mean time between actions during the whole test was measured to be approximately 2 and 3.5 seconds for list and tree layouts respectively. A subject who spent more than 20 seconds on the same screen without any action was removed from the analysis in the final result.

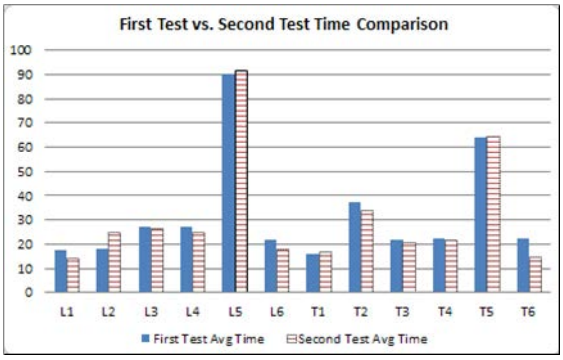

Figure 9: Time Comparison between the First and the Second Testers Showed No Appreciable Difference (X-Axis: Tasks, Y-Axis: Time in Seconds)

Out of the 26 valid test data, 12 subjects took list first test and 14 took tree first test. Comparing the average time and the average number of touches for the same tests done first and second did not, however, yield any significant result showing that second test has any advantage over the first. This check verifies that the counterbalancing was effective and the subjects who did the tree layout test after the list layout test did not complete it faster or with fewer touches than those who did the tree layout test before the list layout test. The comparisons in time and the number of touches are shown in Figures 9 and 10 respectively.

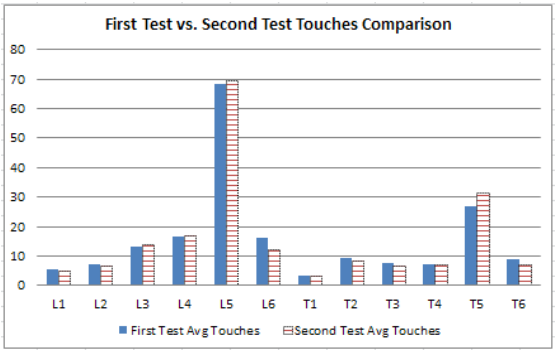

Figure 10: Touches Comparison between the First and the Second Testers Showed No Appreciable Difference (X-Axis: Tasks, Y-Axis: Number of Touches)

Comparing the average number of touches (actions) required to complete each of six different tasks between list and tree showed significant improvement in ERELT based layout. The mean values of number of touches and time for each task are shown in Table 1. In all but Task 2, there was a reduction of about 40% or higher in number of touches required when subjects switched from list based interface to ERELT interface. This anomalous result for Task 2 is discussed in Section 4.7. The comparison data is shown in Figure 11.

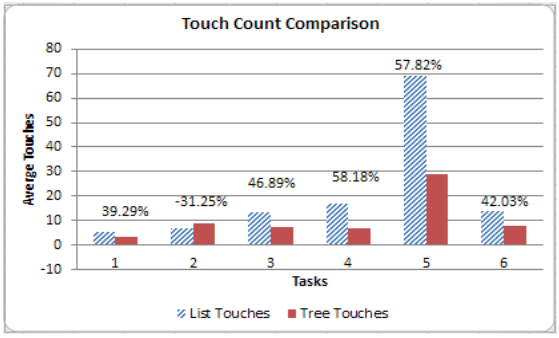

Figure 11: Touch Count Comparison: Percentage Values Show ERELT Improvement Over List Layout

Comparing the average time taken by subjects to complete each of the six tasks in list and tree showed two different results for two different types of tasks (direct path and exploratory). This is discussed in detail in Section 4.7. The comparison data is shown in Figure 12.

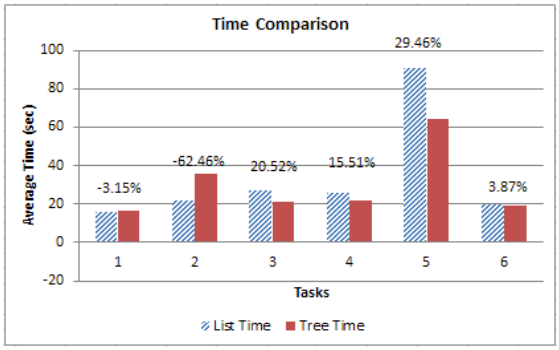

Figure 12: Time Comparison: Percentage Values Show ERELT Improvement Over List Layout

4.6 T-Test

We ran two-tailed paired T-tests to measure the significance of these results. The T-test results are shown in table below.

Table 1: Mean Time and Touches for Each Task with Two-Tailed T-Test Results

Task Type	Task #	Tree Mean	List Mean	p-value	Statistical Significance
Touch Comparison					
Direct Path	**Task 1**	**3.269**	5.385	$p < 0.05$	Yes
	Task 2	8.885	**6.769**	$p < 0.05$	Yes
Exploration & Search	**Task 3**	**7.230**	13.615	$p < 0.05$	Yes
	Task 4	**7.0769**	16.923	$p < 0.05$	Yes
	Task 5	**29.038**	68.846	$p < 0.05$	Yes
Hybrid	**Task 6**	**8.115**	14	$p < 0.05$	Yes
Time Comparison (seconds)					
Direct Path	**Task 1**	16.385	**15.885**	$p = 0.4040 > 0.05$	No
	Task 2	35.615	**21.923**	$p < 0.05$	Yes
Exploration & Search	**Task 3**	**21.308**	26.808	$p < 0.05$	Yes
	Task 4	**22**	26.038	$p = 0.0658 > 0.05$	No
	Task 5	**64.192**	91	$p < 0.05$	Yes
Hybrid	**Task 6**	**19.115**	19.885	$p = 0.3930 > 0.05$	No

The difference in touch count is statistically significant for all the tasks. In time comparison, however, the differences in Task 1, Task 4 and Task 6 are insignificant.

4.7 Interpretation

The average touches and time spent can be interpreted on the basis of tasks in following ways:

1. Tasks 1 and 2 (Direct path task):

In these tasks, the subjects were given the full paths to their targets. Here, ERELT did not have any advantage in time because subjects could just as quickly select paths to their targets in the list. Using ERELT in Task 1 gave subjects a significant advantage in the number of touches because the path was partially visible in the first screen. However, in Task 2, the path to the target was initially hidden due to the branching threshold. Thus the subjects had to scroll to the hidden nodes before a direct path to the target was found and therefore required more touches. Task 2 proved to be the worst in both time and touches for ERELT.

2. Tasks 3, 4 and 5 (Exploration/Search tasks):

The subjects were asked to search for tracks in certain genres, albums, etc, in Tasks 2,4 and 5. ERELT interface was much faster in these tasks because it allowed subjects to quickly move in and

out of various depths of the media library tree. ERELT also displays multiple branches at the same time thus making it easy to search for a tree node. ERELT showed more than 45% improvement in number of touches and 15-30% improvement in time taken to complete these tasks, over the list layout. However, the 15% time difference in Task 4 was found to be statistically insignificant from the T-test. Unlike Task 3, Task 4 requires subjects to navigate back and forth between multiple branches. Thus, this caused large variance in the subject's performance affecting statistical significance.

3. Task 6 (Hybrid):

This task asked subjects to add multiple tracks from a certain genre of a certain language. The first part of the task for the subjects to reach the genre node is similar to direct path tasks. The second part of the task for the subjects to reach various artist/albums in the genre node to add various tracks is similar to exploration task. Since ERELT allows subjects to jump multiple levels up and down, the number of touches was much fewer (10.6 touches per user per task versus 20.9 touches for the list view). Since list view interfaces are advantageous when the path is knows, (especially due to subjects' familiarity with List views), ERELT did not show significant improvement in time.

On average the result shows that ERELT has definite advantage over list-based interfaces in most cases. The fewer touches mean less physical effort from the subjects [18]. Except in the cases of direct and defined path, the ERELT gives comparable or better time performance. We conclude that the ERELT provides a better user interface for exploration/searching tasks. Qualitatively, about 85% of the subjects said they felt tree layout was quicker to work with, as shown in Figure 13.

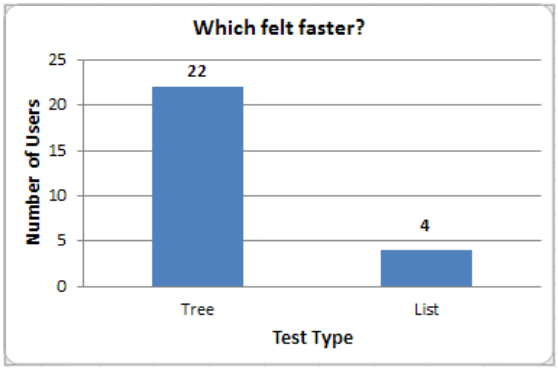

Figure 13: Most Subjects Said Using ERELT Felt Faster

5. CONCLUSIONS

The paper has presented ERELT, a tree visualization technique, designed for displaying and navigating small screens on smartphones. ERELT not only focuses on visual presentation of hierarchical structures, but also contributes on exploration and navigation of such structures through intuitive user interactions, unlike most other tree visualization techniques. The paper addresses two main issues, optimal space usage and rapid navigation of hierarchical structures. Through a user study we have found ERELT to be a faster and less tiring alternative to the traditional list based interface for exploring hierarchical information.

Our immediate future work is to further improve the design based on the feedback from the user study. We will experiment with motion-based gestures for user interactions. Another direction is to explore the capability and usage of ERELT in larger screens such as desktop and laptop PCs, and for visualizing patterns and trends in big data applications.

6. REFERENCES

[1] A. P. Chhetri and K. Zhang, "Modified RELT for Display and Navigation of Large Hierarchy on Handheld Touch-Screen Devices," in *IEEE/ACIS 11th International Conference on Computer*, Shanghai, China, 2012.

[2] H.-J. Schulz and H. Schumann, "Visualizing Graphs - A Generalized View," *Information Visualization,* vol. 9, no. 2, pp. 115-140, 2010.

[3] H.-J. Schulz, S. Hadlak and H. Schumann, "The Design Space of Implicit Hierarchy Visualization: A Survey," *IEEE Transcations on Visualization and Computer Graphics,* vol. 17, no. 4, pp. 393-411, April 2011.

[4] B. Johnson and B. Shneiderman, "Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures," *Proc. IEEE Conf. Visualization (Visualization '91),* pp. 284-291, 1991.

[5] B. Shneiderman, "Tree Visualization with Tree-Maps: 2-d SpaceFilling Approach," *ACM Trans. Graphics,* vol. 11, no. 1, pp. 92-99, 1992.

[6] Y. Kajinaga, T. Itoh, Y. Ikehata and Y. Yamaguchi, "Data Jewelry-Box: A Graphics Showcase for Large-Scale Hierarchical Data Visualization," Technical Report RT0427, IBM Research, 2002.

[7] K. Onak and A. Sidiropoulos, "Circular Partitions with Applications to Visualization and Embeddings," in *Proceedings of the twenty-fourth annual symposium on Computational geometry (SCG '08)*, New York, 2008.

[8] J. Stasko and E. Zhang, "Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations," *Proc. IEEE Symp. Information Visualization (InfoVis '00),* pp. 57-65, 2000.

[9] M. Chuah, "Dynamic Aggregation with Circular Visual Designs," *Proc. IEEE Symp. Information Visualization (InfoVis '98),* pp. 35-43, 1998.

[10] A. Dix, R. Beale and A. Wood, "Architectures to make Simple Visualisations using Simple Systems," in *AVI '00 Proceedings of the working conference on Advanced visual interfaces*, 2000.

[11] J. Hao, K. Zhang and M. L. Huang, "RELT—Visualizing Trees on Mobile Devices," *Proc. Int'l Conf. Visual Information Systems (VISUAL '07),* pp. 344-357, 2007.

[12] J. Hao, C. A. Gabrysch, C. Zhao, Q. Zhu and K. Zhang, "Visualizing and Navigating Hierarchical Information on Mobile User Interfaces," *International Journal of Advanced Intelligence,* vol. 2, no. 1, pp. 81-103, July, 2010.

[13] J. Hao, K. Zhang and C. A. Gabrysch, "Managing Hierarchical Information on Small Screens," *Proc. Joint Int'l Conf. Advances in Data and Web Management (APWeb/WAIM '09),* pp. 429-441, 2009.

[14] J. Yang, M. O. Ward and E. A. Rundensteiner, "InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures," *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.,* pp. 77-84, 2002.

[15] B. Karstens, M. Kreuseler and H. Schumann, "Visualization of Complex Structures on Mobile Handhelds," in *In Proceedings of International Workshop on Mobile Computing*, 2003.

[16] C. Gonzalez, "Does Animation in User Interfaces Improve Decision Making?," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '96)*, New York, 1996.

[17] C. J. Mueller, D. E. Tamir, O. V. Komogortsev and L. Feldman, "Using Designer's Effort for User Interface Evaluation," in *IEEE International Conference on Systems, Man and Cybernetics*, 2009.

[18] L. Feldman, C. J. Mueller, D. Tamir and O. V. Komogortsev, "Usability Testing with Total-Effort Metrics," in *3rd International Symposium on Empirical Software Engineering and Measurement*, 15-16 Oct. 2009.

[19] Adobe, "Adobe Mobile Experience Survey: What Users Want from Media, Finance, Travel & Shopping," October 2010. [Online].