http://dx.doi.org/10.1109/VLHCC.2014.6883017 VL/HCC 2014

# Get Your Directories Right: From Hierarchy Visualization to Hierarchy Manipulation

Rainer Lutz*, Daniel Rausch*, Fabian Beck‡, and Stephan Diehl*
*Department of Computer Science, University of Trier, Germany
‡VISUS, University of Stuttgart, Germany
Email: {lutzr, s4daraus, diehl}@uni-trier.de, fabian.beck@visus.uni-stuttgart.de

***Abstract*—Visual comparison of hierarchies such as directory structures is often considered a passive analysis task. Thus, insights gained from the visualization need to be recorded and applied afterwards. In contrast in this paper, we propose and explore an active visual analytics approach focusing on the manipulation of directory structures in the context of comparison. Two directories including subdirectories and files are presented side by side while links between the two representations connect matching files. Embedded into an elaborate interaction concept, drag-and-drop operations allow the users for interactively modifying the directories. We explored different application scenarios of the approach in a qualitative user study.**

## I. Introduction

Probably the most common form of manipulating a hierarchical structure—applied daily by millions of computer users—is moving files and directories in a file explorer. In *Microsoft Windows*, for example, the user navigates to a directory either by using the directory tree on the left or by clicking through a list of directories in the main panel, to finally select a number of files and/or directories that should be moved. Moving can be done either by drag and drop—provided that the target directory is visible somewhere on screen—or by copy and paste. User interfaces of other operating systems look slightly different, but the process is similar. However, when handling larger directory trees and in particular when comparing or merging different directories, these directory representations show some limitations: First, only a local subset of the directory and file structure is shown—overview is lacking. Second, comparing two directories is not directly supported; the users have to manually arrange two windows side by side. Third, since only a subset of the directory structure is depicted on screen, editing by drag-and-drop operations is often not possible; copy-and-paste operations need to be applied instead.

The goal of the approach presented in this paper is to overcome those limitations of file browsers enabling the comparison and merging of multiple directories. While we do not intend to develop a better general-purpose file browser, we introduce a special-purpose approach focusing on the application scenario of comparing directory structures. We show similarities and differences of two hierarchical directory trees in a scalable visualization. In contrast to existing visual hierarchy comparison techniques, our approach also allows for manipulating the hierarchies. The interaction concept implementing those manipulations uses elaborate drag-and-drop operations for ordering, moving, copying, and merging hierarchy nodes. The approach supports the users in getting their directories right.

## II. Related Work

Our approach combines an existing visual hierarchy comparison technique with a novel interaction concept based on drag-and-drop operations. Hence, approaches that either compare or manipulate hierarchical structures are related. Since hierarchies can be considered as special forms of graphs, interactive graph comparison is also relevant in this context.

**Visual Hierarchy Comparison:** For comparing two hierarchies, Graham and Kennedy [1] identified in their survey on visualizing multiple hierarchies the following paradigms: *edge drawing*, where two hierarchies are juxtaposed and connected by links, *coloring*, where connections are visualized using colors, *animation*, where one hierarchy is smoothly transformed into another, *matrix representation*, where links between hierarchies are encoded as an adjacency matrix, and *agglomeration*, where two hierarchies are merged into one representation. We chose an *edge drawing* approach because it is easy to understand, it explicitly encodes the connections of the two hierarchies, and it is flexible as it works for small changes between the hierarchies as well as for larger ones. While a number of examples for an approach like this can be found in literature [2], [3], [4], [5], our approach is visually similar to the one by Holten and van Wijk [2]: we also use two vertical icicle plots for depicting the two hierarchies in a visually scalable way. In contrast to them, however, we do not apply edge bundling because we try to avoid obfuscating detail information (in particular, the exact source and target of the connections) and keep the approach simple to interpret (edge bundling might confuse unexperienced users).

**Hierarchy Manipulation:** Unlike the visual comparison of hierarchies, the interactive editing of hierarchies has not yet gained much scientific attention. Often, only navigation and adapting the visualization are discussed as interactive features. An example is the interactive comparison of hierarchies on a touch table discussed by Isenberg and Carpendale [6]. A few approaches, however, also cover the manipulation of the underlying hierarchical data. For instance, Beck et al. [7] introduce a tool for interactively transforming the hierarchical structure of software projects. Also for InterRing [8], drag-and-drop hierarchy manipulation operations are described for a single hierarchy. Craig and Kennedy [3] focus on editing relationships between two hierarchical classifications. However, we are not aware of any approach that allows to concurrently edit the hierarchies themselves in the context of hierarchy comparison, for instance, in order to merge the two hierarchies.

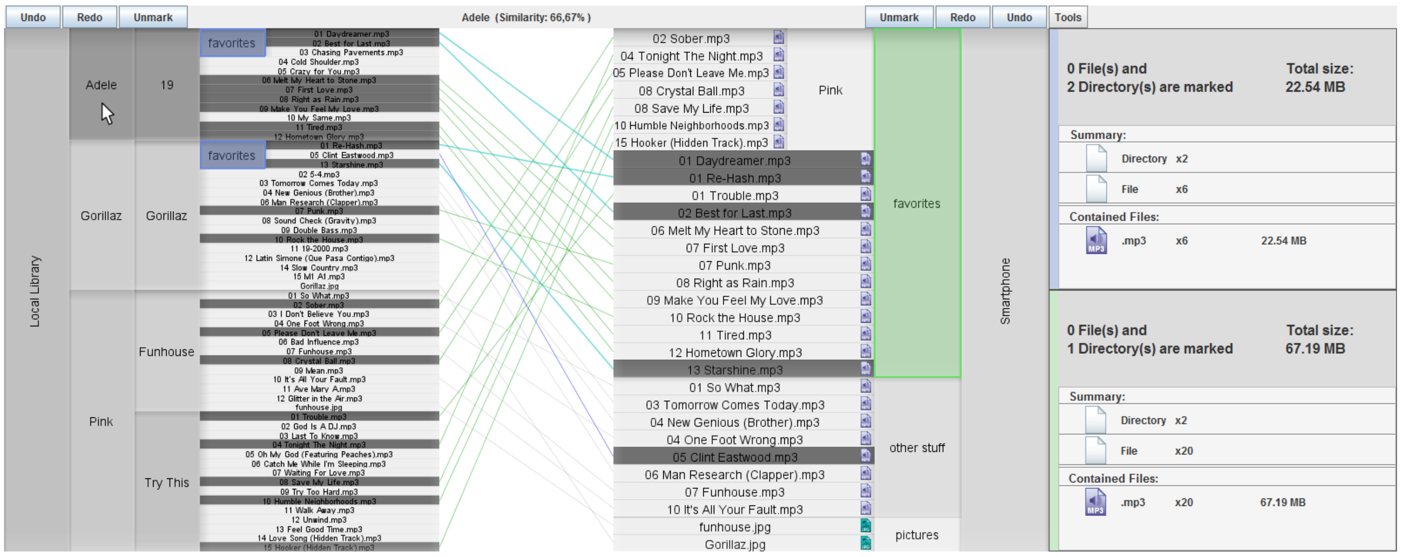

Fig. 1. Comparison of two music collections; favorites directories are selected, directory *Adele* is hovered in the left hierarchy.

**Interactive Graph Comparison:** Different approaches have been proposed for interactively comparing graphs. As well as for comparing hierarchies, most works, however, only focus on interactively displaying similarities and differences, for instance, by overlaying two graphs [9] or by animated transitions [10]. Indeed enabling edit operations, Dadgari and Stuerzlinger [11] suggest a layer-based approach that allows the users to produce a merged result. To this end, nodes from one of the graphs may be accepted or rejected as well as general graph-based modifications can be made. Recently, Koop et al. [12] presented an approach that is able to compare a set of graphs and compute a so called summary graph. Therefore, similar nodes are merged virtually such that users can inspected a combination of all graphs while it is still possible to interactively separate them from each other. Although being applicable in theory, none of these graph-based approaches seem to be very suitable for merging hierarchies: the visualization does not scale to hundreds of nodes and interactions focus on finding equivalent nodes, not on editing containment relations between the nodes.

## III. Interactive Hierarchy Manipulation

Our approach applies a visual hierarchy comparison technique to file systems and extends it for interactively manipulating two directory trees. In particular, the interaction concept supports the following operations:

- to select and compare files and directories from the two directory trees,
- to reorder the contents of a directory,
- to move or copy files and subdirectories from one directory to another,
- to merge the contents of directories, and
- to insert, delete, and rename files and directories.

These operations are realized by simple mouse gestures, mostly based on drag and drop. Since the approach works with a semantic zooming technique and does not require any scrolling, source and target of the drag-and-drop operations are always visible. The combination of visual hierarchy comparison and interactive manipulation supports usage scenarios such as merging or splitting two directory trees and subdividing or restructuring a single one. Although our approach is specialized for the comparison of directories, most of the proposed concepts can be generalized for arbitrary hierarchies.

Our implementation of the approach shown in Figure 1 is named *Directory Comparison and Manipulation Tool* (DCMT); we plan to make the tool publicly available at `http://www.st.uni-trier.de/dcmt/`. Similar to textual diff tools, which are typically not used for editing a single file, DCMT was not developed as an alternative for common file browsers. Instead, it is intended to complement them as soon as users want to compare parts of their file system. To this end, DCMT allows to open two directory trees. An interactive visualization acts as a sandbox environment where changes can be made without being directly applied to the file system. In this sensitive setting, the sandbox prevents accidental changes and improves the performance of the tool (e.g., costly copy operations do not need to be executed immediately). Instead, changes are only realized at the end of the manipulation process on copies of the files.

A directory structure forms a *tree*, which is also denoted as a *hierarchy* in the context of visualization. The *main directory* is the *root node* of the tree, *subdirectories* are the *inner nodes*, and *files* form the *leaf nodes*—we use these pairs of terms interchangeable throughout the paper. For the comparison of directories, it is important to identify equivalent files in two directory structures. While many definitions of equivalence can be applied, we implemented three modes: equivalence based on file names, the size of the files and both, file names and sizes. Please note that a file can have multiple complements in the other hierarchy.

### A. Visual Hierarchy Comparison

As shown in Figure 1, our approach displays two hierarchies that are facing each other with their leaf nodes pointing to the center of the visualization. Both hierarchies are represented as icicle plots, this is, all nodes are drawn as light gray boxes and a complete hierarchy fills a certain rectangular area. The space in between the icicle plots is used to connect equivalent leaf nodes with lines (referred to as *comparison edges*) such that users are able to recall which files occur in both hierarchies. Figure 1 depicts a small music collection that was loaded into our tool in order to compare both the directory structure and the contained files.

The visualization efficiently uses the available screen space and shows both hierarchies independently from each other. Displaying them vertically on a landscape format screen allows a horizontal labeling of leaf nodes. The comparison edges enable users to compare the leaf nodes directly. However, the more similarities between both hierarchies exist, the more comparison edges have to be drawn to connect the equivalent leaf nodes. This may lead to a large number of edge crossings which makes it hard to follow certain lines. Considering this, our tool reduces the number of edge crossings using a barycenter approach as proposed by Holten and van Wijk [2].

## B. Interaction Concept

Our main contribution is not the visualization itself, but rather the features that allow users to manipulate, combine, or split directory trees interactively. In particular, we developed an interaction concept that is based on interactions in common file browsers: it does not overwhelm users with new interaction techniques, but reuses familiar concepts such as drag and drop and extends them towards the new scenario of comparing directory trees. At any time files can be opened with their standard application by using a double click.

*1) Selecting and Comparing Nodes:* The selection of nodes provides details on demand as well as it is a first step towards manipulating the hierarchies. Similar to file browsers, we allow users to select or deselect a single node with a left click and multiple entities by holding the *Ctrl* key while clicking. Thus, the users are free to select any combination of nodes. When a parent directory of an already selected node is clicked, the directory selection replaces the selection of its child nodes, and vice versa. With increasing size of the directory trees, in particular leaf nodes become too small to be clickable. Hence, we need zooming. But we do not want to apply global zooming because sources and targets of comparison edges should be on screen at any time, as well as sources and targets for drag-and-drop operations. Instead, we use local semantic zooming: hovering over a node and using the mouse wheel, its height (together with the heights of its descendants) is increased or decreased; surrounding elements are scaled accordingly without being displaced from screen. By clicking with the mouse wheel, users reset the node to the initial height. Labels and icons are shown if enough space is available.

Fostering the comparison of directory trees, users may select nodes in each hierarchy independently form each other. As depicted in Figure 1, selected nodes in the left hierarchy are colored blue, while those in the right one have a green color. Moreover, the edges that originate from a selected leaf node are highlighted either blue or green depending on which entities were marked, or cyan if both nodes were selected. In Figure 1, the cyan edges quickly reveal that the two selections share four files. The two panels on the right side of the visualization provide further details on the selected nodes, the upper panel with a blue bar for the left hierarchy, the lower panel with a green bar for the right one. When a single leaf node is selected, its path, file size, timestamps, and a preview image or icon are displayed. If multiple nodes are selected like in Figure 1, the number and sizes of marked directories and files is shown as well as an aggregated list of the files by file type.

The selection of nodes helps us further improve the comparison of the two directory trees. Until now, comparing leaf nodes may only be done by following the edges in between the hierarchies. However, inner nodes cannot be compared directly by using these lines. Hence, we also encoded similarities in the hierarchies themselves (referred to as *similarity shading*). In particular, we shade each node of one hierarchy according to its similarity for the selected nodes of the other hierarchy: the higher the similarity, the darker the color. To this end, for each node of the hierarchy, we compute the percentage of its leaf nodes also selected in the other hierarchy. Hence, if a node contains a set of leaf nodes $A$ and a set of leaf nodes $B$ is selected in the other hierarchy, then $|A \cap B|/|A|$ is the similarity value encoded in the node. For instance, in Figure 1 the inner node *favorites* is selected in the right hierarchy, and as a consequence also all of its children. Hence, on the left side all nodes that are equivalent or contain an equivalent node are shaded in gray. For example, the inner node *Adele* shares eight out of twelve files with the selected *favorites* directory on the right side, which can be seen by a dark gray shading. By hovering over a node its exact similarity value is shown in the top bar (here: 66.67%). Since single nodes are either included in the selection or not, leaf nodes are either dark (similarity value 1) or bright (similarity value 0), but nothing in between.

*2) Hierarchy Manipulation:* As a natural and direct way of interacting with visual elements, drag-and-drop operations form the basis of manipulating the hierarchies. To avoid a steep learning curve, we designed these interactions in accordance with common file browsers and in conformance with the users' expectations. Pressing the left mouse button on a node, holding the button, moving the mouse to a different location, and releasing the button forms a drag-and-drop manipulation of the clicked node. If the node is part of a larger selection, the operation refers to all selected nodes. During dragging, we display visual cues in form of red markers as well as icons attached to the mouse pointer indicating what happens if the mouse is released at the current position. Depending on the position, we map the interaction to one of the following six operations, which are illustrated in Figure 2.

**Reorder:** Maybe the least invasive hierarchy manipulation operation is reordering the nodes belonging to a directory: a node is dropped between two nodes within its original directory. This works for single directories or files as well as for several ones belonging to the same directory. While a different order does not make a difference for the file system, it can help grouping similar elements and better see differences between the two hierarchies.

**Move:** If the mouse is released between two nodes of a directory that is different from the source directory, the user performs a move operation of the selected nodes to this directory. Like for reordering, also the exact position in the directory is considered. Selected nodes can be directories and

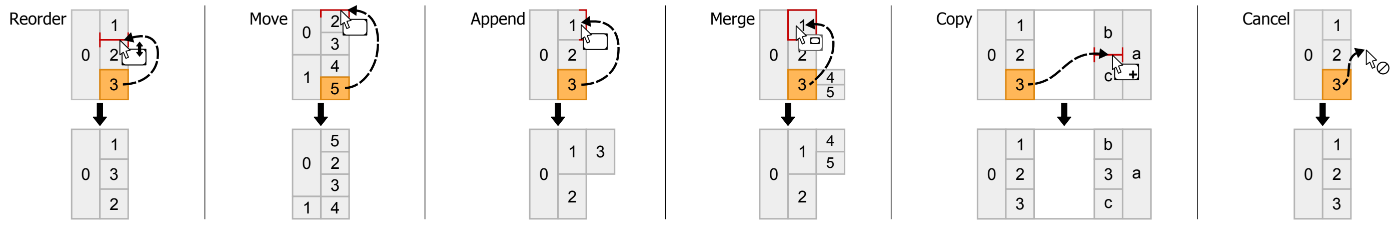

Fig. 2. Drag-and-drop operations for hierarchy manipulation (top) and the resulting changes to the hierarchies (bottom). Red cues mark the drop target.

files from the same or different source directories; their relative order after the move is the same as it was before.

**Append:** Dropping the selected nodes at the inner end of an inner node (i.e., the side closer to the center of the visualization) means appending the selection to this directory. An append operation is equivalent to a move operation to the top position of a directory and, hence, has the same cursor icon. It is specifically necessary for dropping elements into empty directories (cf. Figure 2).

**Merge:** A merge operation is executed when directories are dragged to the center of another directory—the selected directories are resolved and integrated into the target directory. If files are selected as well, these are just appended to the target directory.

**Copy:** When moving nodes from one hierarchy to the other, the above move, append, and merge operations are possible. In contrast, files and directories are not removed from their original location as it is done for operations within the same hierarchy, but copies of them are inserted into the other hierarchy.

**Cancel:** Any drag-and-drop operation may be canceled by releasing the mouse button when the cursor is positioned in the area between the two icicle plots.

Like in the file system, node names within a directory need to be unique. Hence, it is not possible to move or copy nodes to a directory containing already nodes with the same name—the operation is canceled showing a warning. In addition to these drag-and-drop operations, a few other manipulation operations that cannot be easily mapped to mouse movements are available through a context menu: users may rename or remove nodes, create empty directories, and sort nodes automatically. All operations can be undone (and redone, respectively) for the two hierarchies independently by using the buttons above each hierarchy (cf. Fig. 1).

### *C. Example*

To illustrate the use of DCMT, we describe how a hypothetical user of the tool, let us call him Benny, applies the tool to organize his digital music files. He has two small, but partly overlapping music collections—a local library containing all albums on his computer and the songs he likes to have on his smartphone. However, both collections contain songs that are denoted as favorites which Benny would like to organize. In particular, his goal is to compare and restructure the collections with the intention to sort his favorite songs by artist. To this end, he opens both datasets with our tool as depicted in Fig. 1.

As a first step, he inspects the visualized data, decides to keep the left collection, and sort the files on the right side. Also, he identifies and selects three directories named *favorites*—two in the left and a single one in the right hierarchy (cf. Fig. 1). By selecting these directories the visualization shows different information. For instance, the similarity shading on the right side as well as the blue and cyan comparison edges tell Benny which files in the right hierarchy match the selected blue favorites of the left one. There, he quickly identifies an outlier: The second blue *favorites* directory of the left hierarchy contains three files. Two of them (*ReHash, Starshine*) are connected to equivalent files in the green *favorites* folder of the right directory structure. The match of the file *Clint Eastwood* is not located in the green *favorites* folder but rather in a directory called *other stuff*. Moreover, as the *favorites* directory of the right hierarchy is also selected, he can directly see which files are matched with the second blue *favorites* folder on the left side by inspecting its similarity shading and the cyan comparison edges. Thus, while four songs are already included in the green *favorites* directory, Benny moves the fifth one (*Clint Eastwood*) from *other stuff* to *favorites* (cf. Fig. 3, ①).

Vice versa, the similarity shading of the left hierarchy as well as the green and cyan comparison edges provide an overview of which songs are already included in the green *favorites* directory on the right side. As the songs in the left hierarchy are ordered by artist, Benny is able to structure the green *favorites* directory. To this end, he adds two additional subdirectories and names them *Gorillaz* and *Adele*. Then, he moves all songs to the newly created folders according to their position in the left hierarchy. Therefore, he may select the particular artist directory in the left hierarchy such that equivalent songs are shaded gray on the right side and thus can be easily selected and moved (cf. Fig.3, ②). This done, he sorts the green *favorites* directory alphabetically.

Next, he identifies a single outlier in the upper blue *favorites* directory on the left side. The song *Chasing Pavements* is not shaded and, moreover, it has no connection to the right side which indicates that it is currently neither a part of the favorites directory nor of the right hierarchy at all. Hence, he copies that song from the left side to the according subfolder of the green *favorites* directory (cf. Fig. 3, ③). Now that the right hierarchy contains all favorite songs, the *favorites* directories on the left side may be merged with their parents in order to create a well-organized music collection. (cf. Fig. 3, ④).

## IV. Evaluation

Our study was designed as a qualitative, exploratory assessment with a specific focus on testing realistic use cases. We investigated in which way the participants use our tool to compare and reorganize two hierarchies. Thereby, we assessed

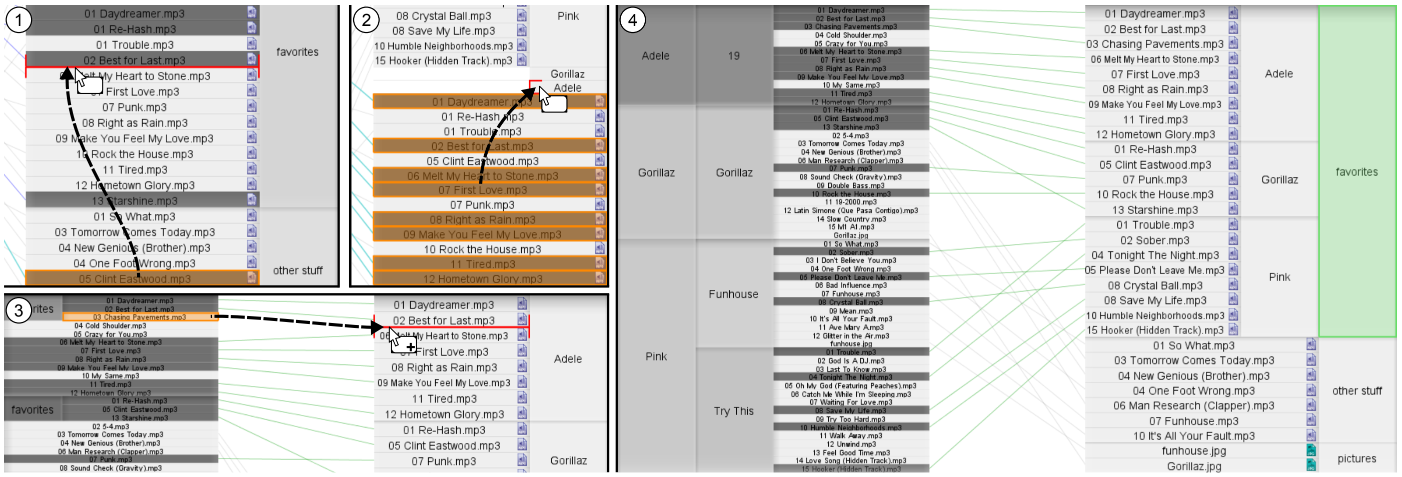

Fig. 3. Process of restructuring two music collections.

the usability of the tool, identified problems, and potential opportunities for improvement. Further goals of the study were to find interesting usage scenarios and strategies as well as to contrast our approach to common files browsers.

## A. Experimental Design

The study consists of two parts: first, the participants were asked to optimize the directory structure of a software system, which is contrasted to a directory structure as proposed by a clustering algorithm; second, the users were free to choose a data set and task that they thought are suitable for our tool. In both parts, participants could freely explore the visualized data using all features of our tool; we imposed neither time constraints nor usage strategies. We captured the screen activities along with the participant's voice such that we were able to explore particular phenomena later on. Two graduate students and two academic researchers took part in our study. On average a single participant spent nearly two hours for the entire session—details are shown in Fig. 4.

At the beginning of each session, we briefly explained the user interface of our tool, provided a short video tutorial that introduces the features of DCMT, and answered further questions from our participants. Then, the two main parts followed, giving the participants the instructions below:

> **Part 1:** *Try to optimize the package structure of the given software project.*

We used the JFtp project as a data set, which is a simple FTP client written in Java consisting of 78 classes and interfaces. To the left side of DCMT, we loaded the package structure of the project (the classes and interfaces organized in directories as done by the original developers), and to the right side a directory structure automatically created by clustering the classes and interfaces of the system (cf. Fig. 5; the clustering is based on structural dependencies as described in previous work [13]). All participants were free to reorganize the package structure in order to produce a result that they found to be an optimized solution. Since the initial package structure of JFtp was quite coarse-grained, all participants were able to further improve it. However, there does not exist an optimal solution for the problem. After Part 1, the participants were asked to propose a dataset that they wanted to investigate and reorganize with DCMT. The task description read as follows:

> **Part 2:** *Please propose a dataset and a task that you think you can use our tool for in this experiment. Please only consider your own data and a manageable task.*

In order to find the reasonable dataset, a longer preparation step was required where the participants browsed their data and discussed several possible tasks with us. The chosen datasets were much larger than the JFtp example and contained up to 200 directories and 4500 files. Some participants preferred to have different source and target hierarchies, for instance, an empty directory or an already ordered music collection on one side and the elements to be sorted on the opposite side. Other participants opened the same dataset twice in order to compare the elements within the same hierarchy and detect duplicates.

After the participants finished Part 2 with DCMT, we asked them to demonstrate how they would solve the same problem with the file explorer of Windows 7. Finally, all participants answered a post-study questionnaire where they assessed the usability of DCMT, reported interesting discoveries in the datasets, and compared our prototype to the Windows Explorer.

## B. Results

Analyzing the results, we investigated the screen captures along with the audio recordings and transcribed important activities and statements to text files. This allowed us to identify interaction strategies, to compare our tool with a common file explorer, and to collect ideas for enhancement.

*1) Interaction Strategies:* Inspired by the categorization for comparing and merging graph-based models that we identified in previous work [14], we grouped the interaction strategies of the participants into four top level categories: *overview*, *navigation*, *comparison*, and *manipulation*. In particular, we followed a lightweight approach using parts of Grounded Theory methodology [15] including open coding. As the personal datasets were much larger than the JFtp example, overview (especially zooming) and navigation strategies played a more

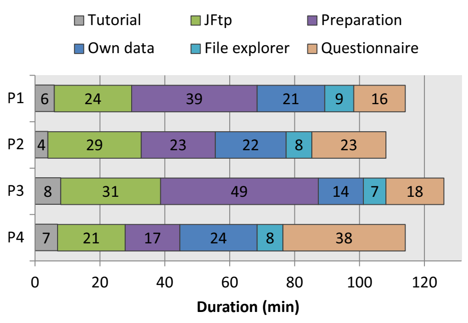

Fig. 4. Time spent by participants P1-P4 for the different parts of our experiment.

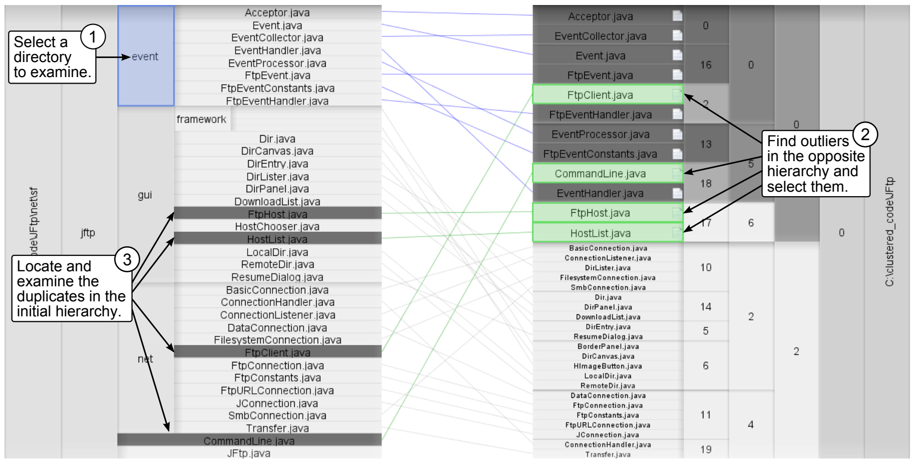
Select a directory to examine. 1
Find outliers in the opposite hierarchy and select them. 2
Locate and examine the duplicates in the initial hierarchy. 3
event
framework
gui
jftp
net
C:\clustered_code\JFtp

Fig. 5. Two step comparison strategy applied by all participants during our evaluation.

important role in Part 2. Next, we present the four top level categories along with typical examples of interaction strategies from both parts of the experiment.

**Overview** comprises strategies to prepare the view on the dataset for a subsequent analysis, e.g., increasing the size of two directories that shall be compared. *Part 1:* After a first glance, all participants found that neither hierarchy was already optimized, i.e., package restructuring was always a combination of both hierarchies. At the beginning of their analysis, the participants roughly inspected and compared both package structures in order to get an overview of the dataset and to decide which hierarchy was supposed hold the merged result at the end. *Part 2:* Due to a larger dataset, the height of the examined directories was often increased such that their content was completely visible. Also, some participants manually reordered the directories inside a branch to reduce edge crossings and facilitate a comparison of the hierarchies.

**Navigation** describes how the participants browsed through the data, in particular, where they started investigating the directory structure and how they proceeded. *Part 1:* The visualized data was browsed in different ways: For instance, smaller packages were inspected first as these seemed to be easier to handle. Some participants started at the top of the visualization and processed the complete set of leaf nodes as if it was a simple list of files. Others traversed the directory structure from the root to the leaves. *Part 2:* In contrast to the JFtp project which was mostly unknown, the participants did not process the their own dataset in a strict way (e.g. package by package), but rather resorted to their background knowledge or chose parts of the data they felt more familiar with. For instance, some participants started to sort their music collection with well-known artists or directories that seemed to be already sorted to some extent.

**Comparison** includes essential strategies to analyze the data in order to find similar directories as well as equal or duplicated files. *Part 1:* All four participants applied the following two step comparison strategy: First, they selected the directory on one side such that equal files and similar folders are shaded in the opposite hierarchy. Second, they identified and selected outliers based on the similarity shadings and inspected if and where these outliers occur in the initial hierarchy by following the comparison edges. Figure 5 shows an example for the JFtp project. Occasionally, when the hierarchical information did not seem to be sufficient in order to optimize the package structure, the participants resorted to semantic information. For instance, they browsed the filenames and looked for similar prefixes or substrings to decide whether these files belong to a certain package or if a new one is needed. Also, they opened particular files to inspect the source code; they searched for similar words, observed the fields of the class, or its inheritance relation. If not contradictory, hierarchical and semantic information often complemented each other. *Part 2:* Our video analysis revealed that comparison edges as well as similarity shadings were mainly used for two reasons: to identify duplicates and to keep track of changes during the merging phase. If the duplicates found were not created on purpose, they were merged in a subsequent step. Since the participants organized their data by copying elements from a messed up to a well structured hierarchy, new comparison edges were created. This allowed the participants to keep track of which directories and files have been successfully integrated into the sorted hierarchy. Moreover, the participants temporarily copied a particular folder to the opposite side in order to compare two directories from the same hierarchy.

**Manipulation** covers such interactions that were intended to modify the hierarchies including move, copy, and merge operations. The information that was gathered through the comparison strategies discussed above allowed the participants to manipulate the hierarchies. *Part 1:* The participants tried to change the hierarchies such that one of them finally contained the optimized result. While some packages were simply adopted or transferred from the opposite hierarchy, others were merged in order to preserve hierarchical information from both sides. Occasionally, the participants also reorganized the package structure based on semantic information or even their own preferences. *Part 2:* Here, the participants did more radical changes. They copied folders and files between the hierarchies, created new directories to introduce an additional categorization, or even flattened parts of the hierarchy if these seemed to be too fine-grained. Irrelevant directories could be identified via the list of file types displayed in the summary panels and elements that have been duplicated on purpose were kept. Also, copies of successfully integrated directories were often deleted from the messed up hierarchy in order to leave more space for the remaining data.

In general, the participants usually pursued the following strategy: After getting an overview of the complete dataset, they started inspecting particular directories or files. During that process they compared the hierarchical information of both sides by using comparison edges and similarity shading. Based on that information they decided whether the compared directories from one hierarchy shall be kept or reconstructed or whether a combination of both sides seemed to be more suitable. If they did not feel confident to make such a decision, they also inspected additional information like the filename, metadata, or even the content of files. Finally, they applied their observations to one of the hierarchies by using DCMT to move, copy, and merge elements between both sides. Where required, they added or deleted directories or files.

*2) DCMT and File Explorers:* In order to assess how well typical file explorers, in this case the file explorer of Windows 7, are suited for the task of comparing and merging directory structures, we observed the participants working with a file browser. Additionally, in the post-study questionnaire, we asked them to describe further experiences when applying such a task. On that basis, we report the following findings.

Although not all of the participants work with the Windows Explorer regularly, they seemed to be more familiar with the different views and the controls of such a file browser. In order to compare and merge two directory structures we allowed the participants to use two different explorer windows such that they were able to move and copy files between two opened directories. In contrast to DCMT, drag-and-drop gestures were less frequently applied and often replaced by copy, cut, and paste activities triggered by keyboard shortcuts. Since not all directories and subdirectories are always visible, the directory tree had to be expanded, folders had to be opened and scrolled. For large and flat directory structures, the participants seemed to prefer scrolling file lists over zooming directories as implemented in DCMT. In particular, one participant found it easier to inspect several files when all of them occupy the same screen space instead of allowing individual zooming levels.

According to the participants, DCMT clearly outperforms the Windows Explorer for the task of comparing directory structures. For instance, they stated that they got a better overview over the complete dataset, in particular, equivalent files could be directly identified through features like comparison edges and similarity shading. Although the Windows Explorer allows to find similar elements by using the search function, this requires several different steps to see additional information and assess whether two files are equivalent. Moreover, with DCMT there is no need to browse directory structures from the root to the leaves as the complete hierarchy is displayed. For the same reason, one participant found it much easier to identify folders that are empty or contain irrelevant data. Also, when copying data with a file explorer, it is directly applied to the hard drive and requires the user to wait a certain time until the process is finished. Using DCMT all modifications to the directory structure are virtual, i.e., elements are not moved or copied on the file system which allows users to experiment with the data without manipulating the original sources.

Furthermore, the merging approach implemented in DCMT, which enables users to combine only the contents of two directories, was appreciated by all participants and should, in their opinion, also be included into the Windows Explorer. However, the version of DCMT the participants used during our experiment did not allow to interactively handle name conflicts when moving or copying directories or files to another location like it may be done in the Windows Explorer. Thus, we recently added such a feature to our tool.

*3) Ideas for Enhancement and Application:* Most participants asked for additional, data-specific similarity metrics that may be used to identify not only equivalent but also similar files. While one participant would have liked to find songs that share the same music genre, another wanted to explore and compare different types of similarity. Moreover, we observed that when the participants marked a directory in one hierarchy, inspected the similarity shadings on the opposite side, and decided to move or copy all of the shaded files to another folder, they had to select them one by one. Hence, a feature like “Select all shaded files” could speed up this process. Furthermore, some participants would have preferred to use a scrolling mechanism instead of zooming.

Also high-level ideas have been proposed on how to use DCMT for organizing data on a file system, for instance, restoring data from a backup after a crash with partial loss of data. While DCMT clearly shows the overlap, using the merging function of a common file browser would be difficult as an overview of both directory trees is missing. Another application is to use DCMT to organize photo collections. Typically, most photos are moved from the memory card of a camera to the computer or an external hard drive in order to archive them. Users want to divide the collection into meaningful categories by merging photos from different sources. Three out of four participants decided to use DCMT to organize their music collections. Although similar to photo collections, for music files the contained metadata seems to be more relevant for finding a decent directory structure. This is why one participant asked for a clustering algorithm that categorizes unsorted files by artist, music genre, or the like.

### C. Validity and Limitations

We conducted a qualitative study with only four participants, which allowed us, on the one hand, to study a realistic scenario as part of a complex experiment, but on the other hand, limits the validity of the findings: first, we are not able to draw any quantifiable conclusions from the experiment; second, individual differences of the participants could have considerably influenced our findings; third, the interpretation of the observed behavior could be biased by the authors. Hence, the results of the study should be considered as preliminary evidence only. Nevertheless, the explorative character of the evaluation allowed us to investigate a broad spectrum of issues, from simple questions of usability to complex usage scenarios.

## V. Application Modes and Tasks

Based on the results of the qualitative evaluation and our experience using DCMT ourselves, we were able to identify different ways data can be loaded and processed in the context of interactive hierarchy comparison and manipulation: different

TABLE I. COMIBINATIONS OF IDENTIFIED APPLICATION MODES AND MANIPULATION TASKS.

| | Merge | Split | Filtered Copy |
|---|---|---|---|
| $X{:}Y$ | $X \cup Y$ | $X \setminus X'$ and $Y \cup X'$ | $X_{\|C_1} \subseteq X$ or $Y_{\|C_2} \subseteq Y$ |
| $X{:}X$ | $X' \cup X''$ | $X \setminus X'$ and $X'$ | $X_{\|C} \subseteq X$ |
| $X{:}\varepsilon$ | $X' \cup X''$ | $X \setminus X'$ and $X'$ | $X_{\|C} \subseteq X$ |

directory structures $X$ and $Y$ can be contrasted ($X{:}Y$ Mode), copies of the same directory structure $X$ may be explored ($X{:}X$ Mode), or one side is left empty such that users are able to create a hierarchy from scratch ($X{:}\varepsilon$ Mode). While the $X{:}Y$ Mode can be considered as the standard mode where a hierarchy comparison is directly applicable, the other two modes do not show any meaningful comparison information at startup. Here, different hierarchies are only created through hierarchy manipulation, whereas the comparison information reflects the progress. In general, we identified three high-level manipulation tasks users may follow in combination with the three modes (Table I specifies all combinations using a pseudo set notation):

**Merge:** Different directory (sub)trees can be merged into a single directory. Thereby, the structure of one of the original trees can be preserved or a new structure might be required. In $X{:}Y$ Mode, usually the different directories $X$ and $Y$ are merged ($X \cup Y$), or at least subdirectories of $X$ and subdirectories of $Y$. In $X{:}X$ Mode and $X{:}\varepsilon$ Mode, the merged directories $X'$ and $X''$ can only be subdirectories of $X$ ($X' \cup X''$); the advantage using DCMT for this task is that $X$ can stay untouched on one side of the diagram while merging is performed on the other side—comparison information documents the current status, in particular, whether already all original files from the two directories are contained in the merged version.

**Split:** Moving a part of the hierarchy structure to another hierarchy means splitting the first. For simplification we assume that we move directory $X'$ being a subdirectory of $X$. In $X{:}Y$ Mode, this means that we somewhere merge $X'$ into $Y$ ($Y \cup X'$), and subtract $X'$ from $X$ ($X \setminus X'$). In contrast, the merging step is not necessary for the two other modes: $X'$ becomes the only content of the second hierarchy. In $X{:}X$ Mode, the effort for reaching that state is larger than in $X{:}\varepsilon$ because the $X \setminus X'$ must be deleted from the second hierarchy.

**Filtered Copy:** When sharing data with others, filtered copies may be used to prepare a dataset or even restrict it to particular directories and files. A common strategy is to start in $X{:}\varepsilon$ Mode opening the data on one side and copying only relevant directories and files to the other hierarchy. This is, the target hierarchy shall only contain data that fulfills a certain constraint $C$. In contrast to this additive approach, $X{:}X$ Mode allows a subtractive approach where one of the hierarchy is reduced until the constraint $C$ is fulfilled. The other hierarchy serves as backup and helps to recall made changes. In $X{:}Y$ Mode two (partly overlapping) hierarchies may be prepared at the same time based on different constraints.

## VI. CONCLUSION

We proposed a new hierarchy manipulation approach for comparing and editing two hierarchies. We specialized the technique and implemented it for manipulating directory trees. Context-sensitive drag-and-drop operations form the core of the interaction concept—nodes can be reordered, moved, copied, or merged. In an explorative study, we observed four users work with the tool in a realistic setting. In general, our participants were able to solve complex tasks by comparing and manipulating directory trees. We identified typical usage strategies and compared these to the strategies applied using a common file explorer. While the participants needed getting used to some uncommon features of our approach such as the zooming concept, the support for comparing directory structures was appreciated and less interactions were necessary for many kinds of operations. Finally, we derived general application modes and manipulation tasks that abstract specific usage scenarios observed in the study.

## REFERENCES

[1] M. Graham and J. Kennedy, “A Survey of Multiple Tree Visualisation,” *Information Visualization*, vol. 9, no. 4, pp. 235–252, 2009.

[2] D. Holten and J. J. van Wijk, “Visual Comparison of Hierarchically Organized Data,” *Computer Graphics Forum*, vol. 27, no. 3, pp. 759–766, 2008.

[3] P. Craig and J. Kennedy, “Concept Relationship Editor: A visual interface to support the assertion of synonymy relationships between taxonomic classifications,” in *Visualization and Data Analysis, San Jose, CA, USA*, vol. 6809, no. 1. SPIE Press, 2008, p. 12.

[4] I. Cruz, W. Sunna, N. Makar, and S. Bathala, “A visual tool for ontology alignment to enable geospatial interoperability,” *Journal of Visual Languages & Computing*, vol. 18, no. 3, pp. 230–254, 2007.

[5] W. N. W. Zainon and P. Calder, “Visualising phylogenetic trees,” in *AUIC ’06: Proceedings of the 7th Australasian User Interface Conference*. Australian Computer Society, Inc., 2006, pp. 145–152.

[6] P. Isenberg and S. Carpendale, “Interactive tree comparison for co-located collaborative information visualization,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 6, pp. 1232–1239, 2007.

[7] M. Beck, J. Trümper, and J. Döllner, “A visual analysis and design tool for planning software reengineerings,” in *VISSOFT ’11: Proceedings of the 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 2011, pp. 1–8.

[8] J. Yang, M. O. Ward, E. A. Rundensteiner, and A. Patro, “InterRing: a visual interface for navigating and manipulating hierarchies,” *Information Visualization*, vol. 2, no. 1, pp. 16–30, 2003.

[9] M. Hascoët and P. Dragicevic, “Visual comparison of document collections using multi-layered graphs,” Laboratoire d’Informatique de Robotique et de Microélectronique de Montpellier (LIRMM), AVIZ (INRIA Saclay - Ile de France), Tech. Rep., Jun. 2011.

[10] L. Zaman, A. Kalra, and W. Stuerzlinger, “DARLS: differencing and merging diagrams using dual view, animation, re-layout, layers and a storyboard,” in *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, ser. CHI EA ’11. ACM, 2011, pp. 1657–1662.

[11] D. Dadgari and W. Stuerzlinger, “Novel user interfaces for diagram versioning and differencing,” in *Proceedings of British HCI 2010*, Sep. 2010.

[12] D. Koop, J. Freire, and C. T. Silva., “Visual summaries for graph collections,” in *Proceedings of Pacific Visualization Symposium*. IEEE, 2012.

[13] F. Beck and S. Diehl, “Evaluating the impact of software evolution on software clustering,” in *WCRE ’10: Proceedings of the 17th Working Conference on Reverse Engineering*. IEEE Computer Society, 2010, pp. 99–108.

[14] R. Lutz, D. Würfel, and S. Diehl, “How humans merge UML-models,” in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2011, pp. 177–186.

[15] A. L. Strauss and J. Corbin, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 2008.