Automatic Expansion of Manual Email Classifications Based on Text Analysis

Enrico Giacoletto and Karl Aberer

School of Computer and Communication Science, EPFL
CH-1015 Lausanne, Switzerland
{enrico.giacolettoroggio,karl.aberer}@epfl.ch

Abstract. The organization of documents is a task that we face as computer users daily. This is particularly true for management of email. Typically email documents are organized in directory structures, which reflect the users' ontology with respect to his daily communication needs. Since users' activities are continuously changing this may render email classifications increasingly inaccurate and manual maintenance is a painstaking task. In this paper we present an approach for integrating user-defined folder structures with classification schemes that have been automatically derived from the email content. This allows to automating the process of evolving and optimizing directory structures without sacrificing knowledge captured in manually created folder structures. A prototype demonstrating the feasibility and utility of our approach has been developed and evaluated. With our approach we address both an important practical problem and provide a relevant study on the application of various techniques for maintaining application specific ontologies.

1 Introduction

The organization of documents is a task that we face as computer users daily. This is particularly true for the management of emails, still the main application of the Internet. Whittaker [19] has written one of the first papers on the issue of email organization. He introduced the concept of "email overload" and discussed – among other issues - why users file their e-mails in folder structures. He identifies a number of reasons: users believe that they will need the emails in the future, users want to clean their inbox but still keep the emails, and users want to postpone the decision about an action to be taken in order to determine the value of the information contained in the emails. He also pointed out that it seems plausible that grouping related emails is considered useful in preserving meaningful context for historical communications and activities and is not simply a strategy to support information search.

Typically email documents are organized in directory structures, where folders and subfolders are used to categorize emails according to various criteria, such as projects, personal relationships, and organizational structures. Therefore these directories reflect the users' ontology with respect to his daily communication needs. Since the communciation is related to continuously changing activities and users cannot foresee

R. Meersman et al. (Eds.): CoopIS/DOA/ODBASE 2003, LNCS 2888, pp. 785–802, 2003.

all kinds of messages arriving in the future this user ontology is anything but stable and can become disorganized quickly. We all know from our daily experience how painful the manual maintenance of email folder structures can be. The user must carefully examine all emails to determine the best possible reorganization according to his current needs. He may want to split folders that have become overpopulated. Or he may want to group all the emails concerning one topic that are scattered over several folders, since at the time of creating the folder structure the emergence of such a topic could not be foreseen. In addition the resulting folder structures are typically far from being optimal, leaving many useless, under-populated folders.

Current email software supports users in automatically classifying emails based on simple criteria, such as sender, time etc., into pre-existing folder structures [1, 2]. However, this does not alleviate the user from first provisioning the necessary folder structures. Also classification of documents based on basic email attributes taken from the header, does not take advantage of the content of the documents during classification. Recent research on ontology development is considering the use of data and text mining techniques in order to derive classification schemes for large document collections [17]. Such an approach appears also to be attractive for addressing the problem of creating email folder structures. However, plainly applying mining tools to email databases in order to create classification schemes, e.g. by applying text clustering techniques [16], does not take into account existing knowledge on the application domain and would render specific knowledge of users in terms of pre-existing folder structure useless.

Therefore we propose in this paper a more differentiated approach. We want to integrate user-defined folder structures with classification schemes that have been automatically derived from the email content. This approach is similar to work that is currently performed in ontology evolution [17], but the profile of the target users is fundamentally different. Whereas in ontology evolution we may expect experts to modify existing ontologies to be shared among large communities in a very controlled and fine-granular manner, probably supported by mining techniques, email users are normally not experts on ontology evolution and integration. Therefore our goal is to automate the process of integrating classification schemes derived by mining techniques with user-defined classifications as far as possible. By analyzing the content of existing email databases we provide classification schemes that are specifically well adapted to the current content of the emails, whereas by retaining the user-provided existing classification by a folder structure we tailor the classification schemes towards the user needs and expectations. This will allow to substantially streamlining the classification schemes being derived from the user and extracted classifications.

The approach we present is based on existing techniques for text content analysis, feature-based clustering and schema integration. The key insight that we want to emphasize is that only by tailoring and combining such techniques in a way that optimally exploits existing knowledge about the structure of the domain it will be possible to have a practical solution for automated ontology generation and evolution. In this sense we not only provide a working solution to a practical problem everyone is facing today, but also a case study on the feasibility and challenges of creating application and personalized ontologies in general.

We will give first in Section 2 an overview of our approach, before we dwelve into the technical details of each of its aspects. In Section 3 we will introduce the various methods for feature extraction we developed. These are essential as they take

specifically advantage of the application domain. In Section 4 we introduce our method for creating folder structures automatically based on feature clustering. In Section 5 we present the algorithm used for integrating computer-generated folder structures with user folder structures. We may view this as a solution to a specific instance of a schema integration problem. In Section 6 we discuss the graphical user interface that has been developed to allow the user to postprocess the automatically generated folder structures. In Section 7 we report on initial evaluation results obtained by providing our solution to real users. In Section 8 we give a short overview on some related work and conclude the paper in Section 9.

2 Overview of the Approach

With our approach we try to combine the best of two worlds: the capability of users to intelligently classify their email and the power of computers to perform computationally intensive content analysis to discover possibly useful content-based classification schemes. An overview of the approach is given in Fig. 1.

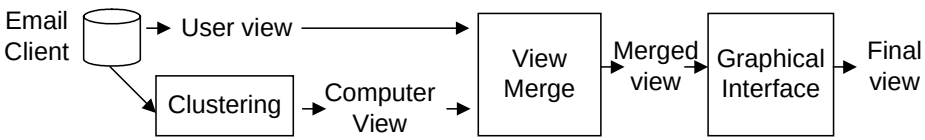


Fig. 1. Proposed solution schema to reorganize user data

Given an email database we assume that a user has already created some existing folder structure which we will call the *user view*. In practice, after some time, this view does no longer match the requirements of the user. Therefore by applying content analysis to the email database we create a new view by providing a computer generated classification of the email data, which we call the *computer view*. The automatic generation of the computer view may be adapated to the needs of the user by controlling which information is considered in the classification process, e.g. topics, dates or headers of emails. The two views may be incompatible. Therefore we need a mechanism to integrate the two views. This is done in a merge step and is a core element of our approach. The view merge algorithm identifies folders with overlapping contents and tries thus to identify semantically identical or related folders. The resulting view is called *merged view*. In general, the merged view will have no longer a tree structure, but be a general lattice, even if both input views had the structure of a tree. The merged view is guaranteed to contain all structural elements of the original user view and to cover the complete email database.

The resulting merged view may still not meet all expectations of the user. Therefore in a final step the user can use a graphical user interface to refine the merged view and thus to produce the final view. This final feedback given by the user is important as only the user can decide what he ultimately wants. The graphical user interface displays the merged view to the user and lets him modify it in accordance with his needs. One issue this graphical user interface has to address is the ability to

deal with lattice structures, rather than trees, as the merged view is in general a lattice. As the merge algorithm summarizes the information of the user and the computer view, for the user it is always possible to retrieve his original view and to remove all the computer's suggestions. He cannot loose work he already invested into classifying emails. In the following we will discuss in more detail the approach taken to realize each of the steps described.

3 Feature Extraction

The automated classification of emails consists of the two steps of feature extraction and classification, which are standard for any content analysis process. Each email has header fields and a body. Among the header fields "To", "Cc", "From", "Date" and "Subject" are taken into account to characterize emails. In the feature extraction process we try to focus already on those features that will be most relevant for the subsequent classification process. Therefore we use different heuristics resulting in the following four feature categories which we have identified as being most relevant for semantically meaningful email classification: one that deals with persons, one that deals with groups of persons exchanging emails over a limited time frame, a so-called email thread, one that deals with subject lines that are related and one that deals with email bodies related over a limited time frame. Each email will be either associated with a specific feature or not. For example, the features from person category will consist of the different persons occurring frequently in emails, and emails will either be related to one of those persons or not. This very selective approach to extracting features is essential in order to optimize the classification schemes that are obtained in the subsequent classification process.

The set of features belonging to each of these feature categories are extracted from the email collection using different methods. We describe these methods in the following shortly.

- *Extraction of person features:* The identification of person features is based on the email address. An email address can contain two fields: the email address itself and the optional name. We use the optional name in order to identify whether the same person uses different email addresses. From this we can create a table relating email addresses with their corresponding person names and thus identify emails belonging to the same person, even if different email addresses were used. Only person names that occur frequently will be used as an email feature.

- *Extraction of email threads:* An email conversation thread consists of a series of replies to a message and the replies to those replies. This is very commonly occuring in email conversations. A thread can also capture complex discussion processes among a group of people whose members may change over time. For extracting threads we examine each email in the database to see if any were sent after the currently examined email involving the sender of the email at the base of the search. There is a time distance limit between every email in a thread list. Only a maximal number of days between two emails are allowed within the same thread. In order to be considered as email feature, a thread must contain more than a minimal number of emails.

- *Extraction of subjects:* Each email has a subject line. Sometimes it is empty, contains a trivial greeting such as “hello” or simply a “Re:”. An email subject containing a greeting is polite, but not very useful in our case. For this reason, we maintain a huge list of “stop words”. These words will not be accepted as features. This list contains French, English, German and Italian words, but it can be adapted to other language(s) being used. A list of all the remaining words encountered in subject lines is created to create the features.
- *Extraction of topics from email bodies:* The extraction of topics from email bodies is based on pairwise evaluation of similarity of the textual content of the email body using a TF-IDF based similarity measure. In order to extract the relevant words from an email body, first text that is not relevant for the evaluation of content–based similarity, like signatures, html tags, webmail signatures and email headers of replies is removed in a pre-processing step. Stopwords for each of the four languages French, English, German and Italian are removed as well. The stopword list was created by using the one found on the Porter’s algorithm web page [10] and the one from [14]. Once all useless words have been removed, the person names and email adresses found in the email body, that have been earlier extracted from the email headers, are removed since person features are covered separately. For the remaining text word vectors are created.

In order to dramatically reduce the effort in identifying related clusters of emails by computing the complete email similarity matrix we exploit a specific characteristic of email texts. The temporal order of writing of emails is known and emails that are temporally distant are less likely related. This idea has been first exploited in [18] for news articles. There exists a great similarity between newspaper articles and emails since both are chronologically ordered documents. The method proposed in [18] allows the identification of email threads in *O(n)* time.

A collection of emails is represented as an ordered set $E = \{e_1, e_2, ..., e_n\}$. The sequence 1, 2,…, n corresponds to the passage of time. Email e_i is older than e_{i+1}. Emails sent on the same day receive an arbitrary order.

The function $sim(e_i, e_j)$ calculates the word-based similarity between an email e_i and an email e_j from the set $comp_{e_i}$, where $comp_{e_i}$ is the set of emails of size *k* considered for comparison.

$$sim(e_i, e_j) = \frac{\sum_{kw} w_{kw}^{e_i} w_{kw}^{e_j}}{\sqrt{\sum_{kw} (w_{kw}^{e_i})^2} \sqrt{\sum_{kw} (w_{kw}^{e_j})^2}} \tag{1}$$

Here, $w_{kw}^{e_i}$ is the weight given to keyword *kw* in email e_i.

$$w_{kw}^{e_i} = \frac{C_{e_i}(kw)}{C_{e_i}} \log \frac{k}{N_k(kw)} g_{kw}^{e_i}, \tag{2}$$

where $C_{e_i}(kw)$ is the frequency of word *kw* in e_i, C_{e_i} is the number of words in e_i and $N_k(kw)$ is the number of emails in $comp_{e_i}$ that contain the word *kw*. $g_{kw}^{e_i} = 1,5$ if $kw \in differential(e_i)$ and $g_{kw}^{e_i} = 1$ otherwise. The function $differential(e_i)$ returns the set of keywords that appear in e_i but do not appear in $comp_{e_i}$.

The original method in [18] was modified in the way of how $comp_{e_i}$ is determined. Rather than considering the *k* last news articles, as in the original method, we consider all emails from from the last *d* days. Taking into account for comparison all emails written in the last *d* days (e.g. *d* = 28) affects the complexity of the algorithm, since the number of emails compared becomes variable. But from the perspective of email analysis it appears more logical to compare an email with all the emails written in a certain period, than with a fixed number of earlier emails. The execution time has not been a practical problem, since the number of emails considered remains much smaller than the size of the email database. As a further optimization the feature extraction algorithm determines in which language every email is written and only emails written in the same language are compared. This can reduce considerably the computation cost for users who receive emails written in different languages.

Email threads correspond then to sequences of emails that are linked by a similarity value $sim(e_i, e_j)$ for $i < j$ that is larger than a given threshold. Each email thread represents a feature.

Once all the features have been determined we eliminate all features and emails that are not useful. A feature appearing only once in an email is not relevant and therefore will be removed. Emails that do not have enough features will also not be taken into account in the following. They would generate non-relevant folders and hurt the result quality. The outlier removal before the start of the clustering and classification process substantially simplifies the subsequent processing.

4 Automatic Folder Generation

A clustering program, Cluto [4], is used to create clusters of emails using the features that have been extracted from the emails. This program takes as input a sparse matrix file whose rows are the data points (in our case email identifiers) and whose columns are the features. The Cluto toolkit was chosen because of its simplicity of use and its flexibility and the fact that it is freely available for research purposes.

The user can choose the number of levels the computer view folder structure should have and the minimal number of folders he wants at each level. The number of clusters per level is (naturally) monotonically increasing with the level of the tree. The clustering tool will then generate at each level in separate runs the required number of clusters. This approach exploits the fact that the clustering tool is using bisective K-Means and is thus creating clusters by successive (binary) splitting of the data sets. More precisely the tool proceeds by first splitting the data sets into two subsets with the closest features, then selecting one of the sets created in this way for further splitting, and so on till the desired number of clusters is reached. So if, for example, a clustering for the first level of the folder hierarchy has been created, a

second execution of the clustering will again first create the same clusters and then continue to refine these clusters further, thus creating the second level partitioning. We give a simple example illustrating of how the construction of the folder structure works in Fig. 2.

email ID	clusters level 1	clusters level 2
1	1	1
2	1	1
3	1	2
4	2	3
5	2	3

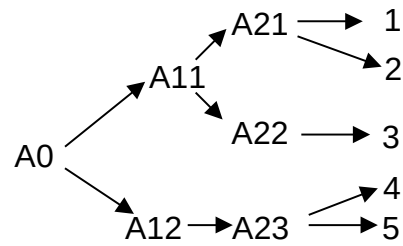


Fig. 2. Creation of folder structure through clustering

The optimal number of clusters to be generated is difficult to determine since it depends both on the available data and the user intentions. If the clustering program generates too many folders, the potential problem is that the subsequent merge algorithm, which will be described in the next section, will create too many folders in the merged view and related emails run the risk of being scattered over multiple folders. Also result visualization and comprehension becomes difficult. If the clustering program does not generate a sufficient number of folders some folders will contain emails on different topics that do not belong together.

The following approach has been used to achieve an optimal number of folders and still to allow the user to specify a minimal number of folders to be generated. The minimal number of folders required by the user is the number of clusters that the clustering program Cluto will have to find among for the email collection. These email clusters will become folders. There is no warranty that all the emails in a folder are related, as we want them to be. For this reason there is a further step, called group outlier removal, which will check every folder and determine if all its emails are related. If not, it will create new folders until all the emails contained in one folder are related. This approach will produce the minimal number of folders needed to generate a meaningful view.

We considered two methods to detect outlier emails within a generated folder. In the first approach we try to identify a representative email that shares with each other email in that folder at least one feature. We may consider this as testing whether the cluster contains a central element. In the second approach we test whether each email in the folder shares at least one feature with some other email in the folder. We may consider this as testing whether the cluster is connected. In both cases we exclude features related to the user's email address, as they would in general relate all emails of a folder. The first approach generates more folders and decreases the chances that one folder contains more than one topic. For this reason this is the method we have chosen for our implementation. Figure 3 illustrates the group outlier removal process. Group outlier removal proved to be very effective in order to dramatically improve the quality of the merged views.

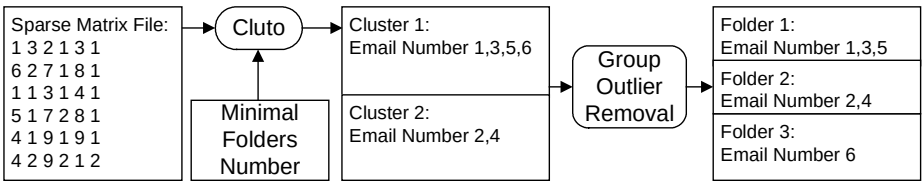


Fig. 3. Email 6 is an outlier in folder 1 as it cannot share a common feature with a central email (e.g. 1, 3). The group outlier removal creates a separate folder for it, if it is based on the first method presented

5 Folder Merge Algorithm

The folder merge algorithm takes two trees as input (the user view and the computer view) and produces a directed graph called merged view. First the merge algorithms needs to be able to determine when two folders are to be considered and thus should be merged. For this we introduce a similarity measure for folders. Then we discuss the different operations that can be performed to merge folders depending on their similarity. Finally we present the global algorithm and its properties. Note that even though we present this algorithm for the context of email folders it is generally applicable to any hierarchical schema used for data classifications.

5.1 Folder Similarity Measure

We view a folder structure as a tree where internal nodes correspond to folders and leaf nodes correspond to emails. For a folder *A* we denote by *leaf(A)* the set of all direct and indirect leaves that it contains. Then the Shared Leaves Ration (*SLR*) is the measure used to determine the similarity of the content of two folders. This measure will be used to identify which folders should be compared and what actions are taken based on the comparison. The Shared Leaves Ratio comparing folders *A* and *B* is defined as

$$SLR_A(B) = \frac{\left|leaf(A) \cap leaf(B)\right|}{\left|leaf(A)\right|} \tag{3}$$

It is important to note that *SLR* is a non-symmetric relationship. If

$$\left|leaf(A)\right| \le \left|leaf(B)\right|$$

then

$$SLR_B(A) \le SLR_A(B)$$

with equality only if

$$\left|leaf(A)\right| = \left|leaf(B)\right| .$$

5.2 Basic Operations

Different merge operations can be applied after two folders from the user and the computer view have been compared. We introduce first the basic merge operations and then the merge algorithm that applies these operations in a proper order to merge two folder structures.

Basic operations always involve a folder *A* from one of the two input trees. The list G_A contains all the folders *C* such that folder *C* and *A* have some leaves in common (i.e. contain some common emails). If G_A is non empty, the merge algorithm will try to combine folders contained in list G_A with the folder *A*, otherwise folder A will no be modified and be considered as merged. The merge algorithm will be designed such that it always guarantees that the condition $|leaf(A)| \leq |leaf(C)|$ is satisfied (see algorithm *FolderSelection* in Section 5.3). Therefore when introducing the basic operations we will assume that this condition holds.

Specialization
If, after comparison, it turns out that a folder *C* in G_A is contained in *A* then *C* can appear as a subfolder of *A* in the merged folder structure. This situation occurs if

$$SLR_A(C) = 1, C \in G_A \tag{4}$$

An example of specialization is given in Fig. 4. Creating subfolders through specialization is particularly interesting for partitioning large folders in the original folder structure into smaller folders.

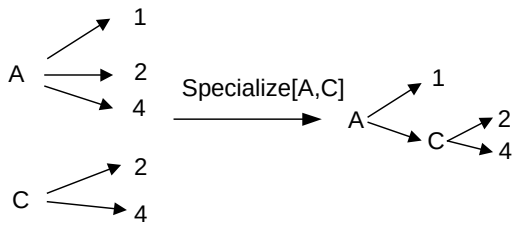


Fig. 4. Result of the specialize operation

Specialization with a Shared Folder
This operation is executed when folder *A* and a folder *C* in G_A have only some leaves in common, but neither is contained in the other. This operation creates a new folder to contain the common emails. The new folder becomes a subfolder of both folders *A* and *C* and thus in general the folder structure generated by the merge algorithm will be a directed graph. This operation occurs if:

$$0 < SLR_A(C) < 1, C \in G_A \tag{5}$$

An example of specialization is given in Fig. 5.

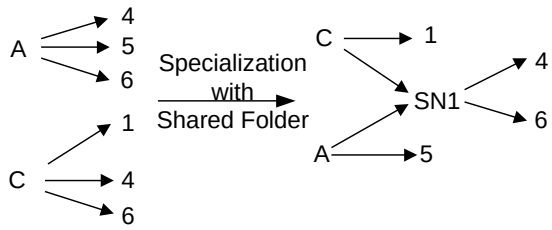


Fig. 5. Result of the specialization with shared folder operation

This operation is useful in dealing with emails that belong to more than one topic. It is also useful to correct classification errors. It happens quite often that an email is classified into a wrong folder. Such an email will appear in the wrong folder in the user view, but it will be regrouped with related emails in the computer view. After performing this operation, the problematic email will appear both in the wrong and right folder. Later, this email will attract the user's attention when he is inspecting the folder structure with the graphical user interface and he will be able to decide what to do. Either he or the computer did a classification mistake and the email should appear in only one folder, or it is of interest to have the email in two folders, because it is related to more than one topic.

If G_A contains more than one folder, folder *A* shares leaves with more than one folder *C*. In this case, the basic operation of specialization with a shared folder is executed once with every folder in G_A, but folder *A* is inserted only once into the merged graph.

Insertion of the Folder in the Merged Graph

If none of the two basic specialization operations can be executed, i.e. if G_A is empty, the folder *A* is inserted in the merged graph without modifications. The leaves of folder *A* are as well inserted into the merged graph.

5.3 Merge Algorithm

We give a general overview of the merge algorithm that relies on the basic operations. For initialization the computer and user view are merged into a common tree. This step is performed by merging the two root nodes of the two trees into a common root node resulting in the unified input tree.

After initialization the merge algorithm can be divided into three steps which are repeatedly executed as illustrated in Fig. 6. The first step called *TreeTraverseStrategy* ensures that the unified input tree is traversed in the correct order. It maintains a folder list called *PossibleBaseFolder*, which contains the list of folders whose sons must be merged. *PossibleBaseFolder* is a FIFO queue that determines the folder *currentRoot*. Initially *PossibleBaseFolder* contains the common root node of the unified tree. This folder is passed to the next step of the algorithm.

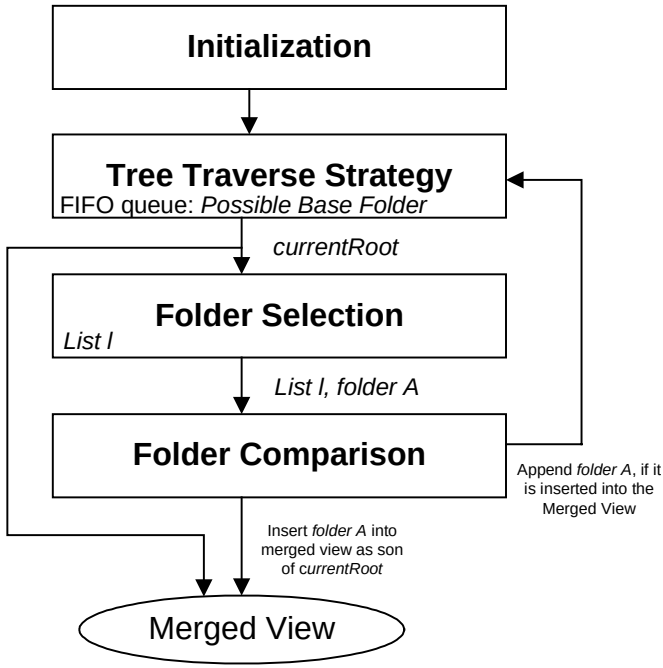


Fig. 6. Merge algorithm structure

The second step called *FolderSelection* determines a folder list *l*. This list contains the sons of the folder *currentRoot*. Once the list is determined, it is sorted in increasing order according to the number of leaves each folder contains. The first element of this list will then successively be removed from list *l* and passed to the next step of the algorithm, along with the remaining list *l*. The folder passed successively to the next step of the algorithm part will be called folder *A*.

```
FolderSelection(currentRoot)
          l = { A | A is a son of currentRoot};
          Sort l in increasing order according to |leaf(A)|;
          while l not empty
             A := first element of l;
             Drop first element of l;
             FolderComparison(l, A)
          endwhile;
```

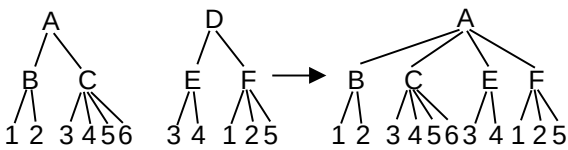
The third and last step, called *FolderComparison*, derives the list G_A from *l*. This is done by calculating the *SLR* of all vertices in list *l* with folder *A* and including those folders in *l* that have a *SLR* greater than 0 into G_A. If G_A is empty, the folder will be inserted into the merged graph directly. If there is an element in G_A that satisfies the condition for the specialization or specialization with a shared folder operation, then the operation is executed. Every time when the algorithm adds a folder to the merged view by executing a basic operation, this folder is also added to the FIFO queue

PossibleBaseFolder. The detailed folder comparison algorithm is given in the following

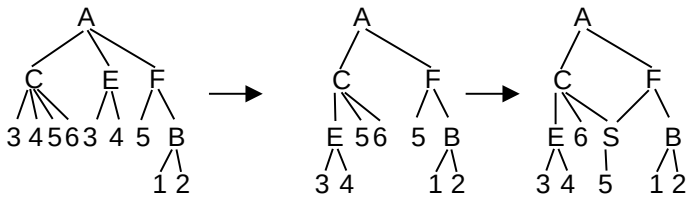
```
FolderComparison(l, A)
    Calculate SLR_A(C) for each folder C ∈ l;

    CloseFolders = { C | C ∈ l, SLR_A(C) > 0 };

    Sort CloseFolders in decreasing order according to SLR_A(C);
    ClosestFolder = first element of CloseFolders;
    nrSharedLeaves = |leaf(A) ∩ leaf(ClosestFolder)|;
    if SLR_A(ClosestFolder) > inclusionThreshold
        then  specialize(A, closestFolder)
        elseif SLR_A(ClosestFolder) > 0 and
                    nrSharedLeaves ⩾ nrSharedLeavesThreshold
            then specializeWSharedFolder(A, CloseFolders);
                   insertFolderIntoMergedGraph(A, currentRoot)
            else insertFolderIntoMergedGraph(A, currentRoot);
```

The algorithm terminates when *PossibleBaseFolder* is empty while executing *TreeTraversalStrategy*. This algorithm does not loose any information that is contained in the user view. All folders from the user view will be retained in the merged view. Also no emails can be lost since all emails contained in a folder of the user view will show up as elements of the respective folder retained in the merged view.

We demonstrate the execution of the merge algorithm by a simple example. Let the following two folder structures be given as input.



The first step is to create a unified tree with a common root. At the beginning *CurrentRoot* is set to *{A}* *PossibleBaseFolder* is empty. The folder selection algorithm produces initially a list *l = {B, E, F, C}*. Processing folder *B* produces the first graph in the following figure. The following steps are related to processing the remaining elements in *A*.



Once the merge algorithm termiantes, the output must still be post processed. Folders with no leaves and containing only a single folder will be removed, unless these folders are not user folders.

5.4 Label Creation

During the merge processes new folders are created by the algorithm. For these folders new labels have to be created. Creating intelligible labels is an extremely important and equally difficult problem. The labels are generated using the features of the emails in the folder. We chose to use both person and subject related features. We developed several rules to generate from terms found in the emails labels enriched by additional information extracted from the structure of the email. The following rules were applied to that respect:

- email f *name* a *subject*: a folder with this label contains essentially emails from a person with *name*. The most frequent terms related to a subject are indicated.
- a *subject*: a folder with this label contains emails essentially related to a specific subject, which is the most frequently observed feature.
- g-email(*number of person in the email-list*) *name1 name2 email-list* a *subject words*: a folder with this label contains essentially emails which had always the same recipient list *email-list*. The first two members of the email list are mentioned.
- e received w *name* a *words*: a folder with this label contains essentially emails that have been received from a person with *name*. Frequently emails of this type are sent via a distribution list.
- discussion w *name* a *subject*: a folder with this label contains emails where a discussion is carried on with person *name*. The person whose name is included into the label is involved as person feature in most of the folders or emails, both in the from field and in the recipient list.

6 Graphical User Interface

In general, the merge algorithm results in a directed graph. This graph contains new access points to email either created through folders integrated from the computer view or created through the specialization with new folder operation. Since the resulting folder structure is a directed graph it cannot be directly displayed with standard tools for tree-like directory structures. A special user interface was thus developed to allow the user to visualize the generated folder structure, and more importantly to manipulate it according to his needs. The user interface also allows to filtering the merged view in order to focus the users' attention on specific aspects of the folder structure. Thus the user interface provides a tool that allows the user to efficiently transform the merged view into the desired final user view.

The graphical user interface for postprocessing the merged view is an essential part of the solution we propose. Ultimately, only the user is able to decide whether certain folders created by the preceding process are semantically meaningful. All the steps for the automatic reorganization of folder structures presented earlier are designed in

order to produce potentially useful results, but the final decision about their utility has to be left to the user.

The most challenging requirement when designing the graphical user interface was to provide the user the possibility to explore the different parents of a folder and at the same time exhibit the structure of the original user view. In order to help the user to navigate easily in the result, tools have been added that aid in quickly understanding the automatically created folder structure and accepting or rejecting changes swiftly. For doing that two modes of browsing are supported, namely browsing vertically along the paths of the tree hierarchy and browsing horizontally among the different fathers of a shared folder. Folders from the merged view which should be examined by the user are specially marked (with the letter "M" as it is shown in Fig. 7). Marked folders that have been examined and processed by the user are unmarked.

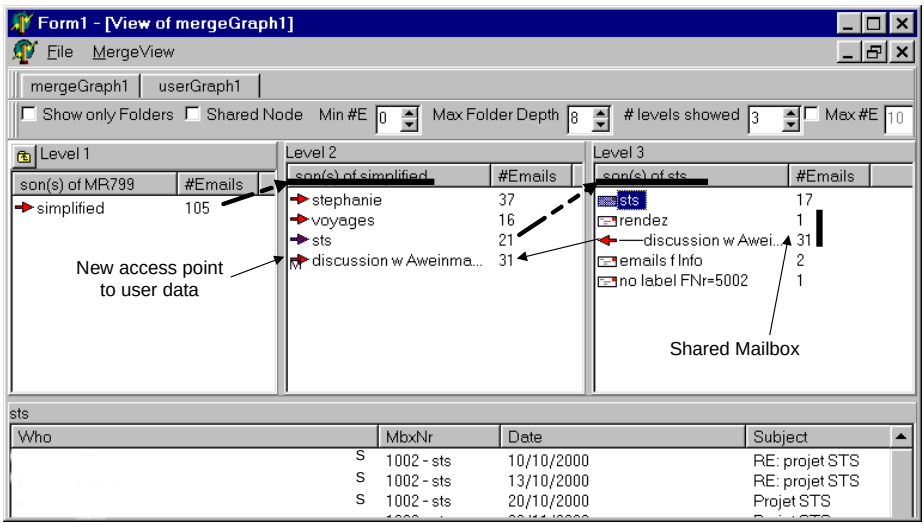


Fig. 7. This is a screenshot of the implementation of the graphical user interface. It shows how folder structures are displayed and how the different characteristics described in this section are implemented. The letter "M" near the folder "discussion w Aweinma…" indicates to the user that he should start to examine the merged view by looking at this folder and thus to focus his attention.

With respect to editing of the folder structures the following rules have to be observed. (1) It is only possible to remove a folder if it has no parents left and (2) if a folder that is shared among multiple folders is removed, it is only removed from one of the parent folders. The second rule allows to reorganizing shared folders such that a more tree-like structure can be achieved. With respect to filtering the user can specify to see only folders that contain a maximal or minimal number of emails. This is useful to eliminate potentially useless folders and thus to reduce the complexity of the folder structure.

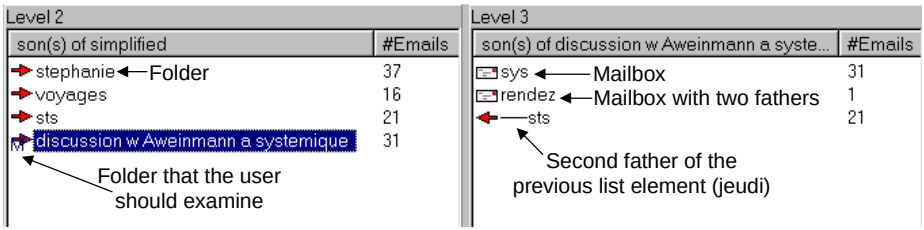


Fig. 8. This figure shows how the graphical user interface displays folders having more than one parent

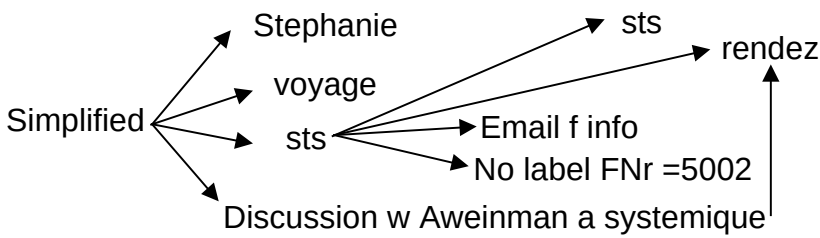


Fig. 9. This figure shows the folder structure of the merged view displayed in Fig. 8. It is interesting to observe of how the folder "rendez", that has two parents, is displayed.

7 Evaluation

The approach described in this paper has been fully implemented. To connect the various steps of the overall process, including feature extraction from emails, processing by the clustering tool and display at the graphical user interface a set of perl scripts was used that communicate through files. The graphical user interface has been implemented in Delphi. For mailboxes the Eudora format has been used.

For evaluation different user tests have been performed. Two types of questions have been addressed in the evaluation: the evaluation of the quality of the merged view and the evaluation of the usability of the graphical user interface. A secondary issue that we evaluated was the computational performance of the implementation of the algorithms. The results of the usability test were reflected in several redesigns of the graphical user interface. In the following we focus on the test concerning the generation and quality of the merged view. We report the results from one representative test.

A user provided his mailbox containing 3466 emails organized into 4 folders. The user then requested the tool to produce a merged view on two levels with 5 folders on the first level and 20 folders on the second level. It took approximately 26 minutes to produce the merged view on a Pentium PC with 700Mhz. After the creation of the merged view the user inspected 15 of the new folders. 11 out of the 15 folders were judged to be meaningful since they concerned a specific topic. The user remembered that sometime he classified his emails without much consideration, but he was astonished to see how many topics were spread over different mailboxes. Most of the time topics were split among two folders, but one topic even had been split among

three folders. The 4 folders that were considered meaningless had two kinds of problems. The most frequent problem (also in other experiments) has been folders containing unrelated emails. This typically results from using person-related features from the email. This indicates a possible improvement in the feature extraction process. The second problem was related to the intelligibility of the produced folder structure. As it turns out it is difficult for a user to well understand the folder structure, in particular in the presence of shared folders, when the depth of the folder structure is more than one.

From this and the other tests made (a total of 9) we can thus draw the following conclusions. The different tests show that a majority of the folders and mailboxes produced by the proposed solution are meaningful. An average of 50 to 70% of meaningful folders is achieved when the number of clusters from which the computer view is generated is reasonable. It is difficult to say in general what a reasonable value would be as this parameter depends on the characteristics of the user mailbox and the number of preexisting folders. But experience showed that this parameter should be chosen such that the number of folders in the computer view should be approximately the number of folders in the user view, unless the user view does not contain too many small folders that the user wants to group together. Increasing the minimal number of folders to be generated increases the probability of creating meaningful folders, but it decreases the probability of finding all the emails related to a topic in the same folder. Once a user has made the necessary effort to understand the graphical user interface, the comprehension of a merged view of depth one is straightforward. Some parts of the merged view become difficult to understand if the depth of the user or the computer view is greater than one. There are two possible solutions to improve the result intelligibility. Either the display at the graphical used interface is changed or the merge algorithm is modified such that complex folder structures are avoided while maintaining the user information retention property.

8 Related Work

The application of text classification techniques to email documents has been widely studied in the literature. We do not give a complete overview on this field, but just mention some typical examples. We can characterize this work along two dimensions: the techniques used and the specific application targetted. As for the techniques that have been considered in the literature we find rule induction [1], word-based TF-IDF weighting [15], naïve Baysian text classification [8, 11, 12], and hierarchical clustering [6], the latter being the method also we were using. In most cases emails are considered as unstructured text, not taking advantage of the specific email structure during feature extraction. A notable exception is [6], who use similar feature extraction methods as we do. As for the scope of the different approaches they are more limited than our approach, as they focus on a specific task. The following standard tasks can be identified:

- automatic creation of folder structures, e.g. [6]
- automatic classification by filtering of email messages, e.g. [12]
- filtering of email messages into user defined folders, e.g. [1]
- spam detection, e.g. [8]

None of these approaches aims at reorganizing user-defined folder structures as we do. Either no user-defined folders are used at all and classifications are automatically generated only, or the preexisting folder structure is the target of an email filtering method. On the other hand these works provide a rich set of methods that could be used as alternative methods to the one we have applied in the computer view generation. Our focus was not so much to make use of sophisticated text classification techniques, but to use existing techniques in a way that took most advantage of the specific problem structure at hand.

The merge technique that we applied is essentially based on ideas that have already been introduced in the very first works on database schema integration [7]. The approach of integrating techniques for document analysis, for schema integration and user interfaces can be seen in line with current developments on ontology engineering tools [3]. Similarly as our work, these tools have been recently extended by introducing document analysis methods, as in [5], to enable semi-automatic development of ontologies.

9 Conclusions

In this paper we have introduced an approach to evolve user-defined email folder structures taking advantage of text content analysis techniques. One challenge was to completely preserve existing knowledge of users while at the same time apply modifications obtained from email content analysis. In this sense the method bridges the gap on earlier work on email filtering (relying on user-defined folder structures only) and email classification (relying on computer generated folder structures only). Another challenge was to address all the phases of the process starting from email classification, to folder merging and interactive postprocessing of the result by the user, while taking in each phase advantage of the specific knowledge available on the email domain. The approach has been implemented and evaluated.

The approach is besides addressing an important practical problem also a relevant study on the application and combination of various techniques for creating and maintaining user/application specific ontologies. Many of the observations and techniques would naturally generalize to onotology management in more general settings.

References

1. Cohen, W. W.: Fast Effective Rule Induction. Proceedings of the Twelfth International Conference on Machine Learning (ICML), Tahoe City, CA, USA, July 9–12, 1995, Morgan Kaufmann, pp. 115–123, 1995.
2. Crawford, E., Kay, J., McCreath, E.: IEMS – The Intelligent Email Sorter, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8–12, 2002, Morgan Kaufmna, pp. 83–90, 2002.
3. Duineveld, A. J., Stoter, R., Weiden, M. R., Kenepa, B., Benjamins, V. R.: Wondertools? A comparative study of ontological engineering tools, Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management, Voyager Inn, Banff, Alberta, Canada, October 16–21, 1999.

4. Karypis, G.: Cluto A Clustering Toolkit manual, University of Minnesota, Department of Computer Science Minneapolis, MN 55455, related papers are available at http://www.cs.umn.edu/~karypis.
5. Maedche, A., Staab, S.: Semi-automatic Engineering of Ontologies from Text. Proceedings of the Twelfth International Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, July 6–8, 2000.
6. Manco, G., Masciari, E., Ruffolo, M., Tagarelli, A.: Towards an Adaptive Mail Classifier, Proceedings of Tecniche di Intelligenza Artificiale per la ricerca di informazione sul Web (AIIA), Siena, Italy, Septemmber 11–13, 2002.
7. Motro, A., Buneman, P.: Constructing Superviews, Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, April 29–May 1, 1981, ACM, New York, pp. 56–64, 1981.
8. Pantel, P. Lin, D.: SpamCop – A Spam Classification & Organization Program. AAAI-98 Workshop on Learning for Text Categorization, pp. 95–98, 1998.
9. Pazzani, M.J.; Representation of electronic mail filtering profiles: a user study. Proceedings of the 2000 International Conference on Intelligent User Interfaces, January 9–12, 2000, New Orleans, LA, USA, ACM, New York, pp. 202–206, 2000.
10. Porter, M.: Porter's algorithm implementation official page, http://www.tartarus.org/~martin/porterstemmer/
11. Provost, J.: Naïve-Bayes vs. Rule-Learning in Classification of Email. The University of Texas at Austin, Artificial Intelligence Lab. Technical Report AI-TR-99-284, 1999.
12. Rennie, J.: ifile: An Application of Machine Learning to Mail Filtering. Proceedings of the KDD-2000 Workshop on Text Mining, Boston, USA, August 20, 2000.
13. Rohall, S: Reinventing Email. CSCW 2002 Workshop: Redesigning Email for the 21st Century, 2002.
14. Savoy J.: Report on CLEF-2001 Experiments. In C. Peters (Ed.), Results of the CLEF-2001, cross-language system evaluation campaign, (pp. 11–19). Sophia-Antipolis: ERCIM, http://www.unine.ch/info/clef, 2001,
15. Segal, R., Kephart, M.: MailCat: An intelligent assistant for organizing e-mail. Proceedings of the Third Annual Conference on Autonomous Agents, May 1–5, 1999, Seattle, WA, USA. ACM, pp. 276–282, 1999.
16. Steinbach, M., Karypis, G., Kumar, V.: A Comparison of Document Clustering Techniques, Proceedings of the KDD-2000 Workshop on Text Mining, Boston, USA, August 20, 2000.
17. Sure, Y., Angele, J., Staab, S.: OntoEdit: Guiding Ontology Development by Methodology and Inferencing. Proceedings of the Confederated International Conferences DOA, CoopIS and ODBASE 2002 Irvine, California, USA, October 30–November 1, 2002, LNCS 2519, Springer, pp. 1205–1222, 2002.
18. Uramoto, N., Takeda, K.: A Method for Relating Multiple Newspaper Articles by Using Graphs, and Its Application to Webcasting. Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, August 10–14, 1998, Université de Montréal, Quebec, Canada. ACL / Morgan Kaufmann, pp. 1307–13, 1998.
19. Whittaker, S., Sidner, C.: Email Overload: exploring personal information management of email. Proceedings of the Conference on Human Factors in Computing Systems (CHI'96), April 13–18, 1996, Vancouver, British Columbia, Canada, pp. 276–283, 1996.