CEM – A Conceptual Email Manager

Richard Cole[1], Gerd Stumme[2]

[1] School of Information Technology, Griffith University, Gold Coast Campus, PMB 50, Gold Coast Mail Centre QLD 9726, Australia; r.cole@gu.edu.au
[2] Technische Universität Darmstadt, Fachbereich Mathematik, Schloßgartenstr. 7, D–64289 Darmstadt, Germany; stumme@mathematik.tu-darmstadt.de

Abstract. CEM is an email management system which stores its email in a concept lattice rather than in the usual tree structure. By using such a conceptual multi-hierarchy, the system provides more flexibility in retrieving stored emails. The paper presents the underlying mathematical structures, discusses requirements for their maintenance and presents their implementation.

1 Motivation

The way standard email management systems store mails is directly derived from the tree structure of file management systems. This has the advantage that trees have a simple structure which can easily be explained to novice users. The disadvantage is that at the moment of storing an email the user already has to foresee the way she is going to retrieve the mail later. The tree structure forces her to decide at that moment which criteria to consider as primary and which as secondary. For instance, when storing an email regarding the organization of a conference, one has to decide whether to organize one's directories like `mineau/iccs2000/program_commitee` or like `conferences/iccs/iccs2000/organisation/mineau`. This problem arises especially if a user cooperates with overlapping communities on different topics.

In this paper, we present the *Conceptual Email Manager CEM*. It uses a formal context as its structure for storing email rather than a tree. This allows the user to retrieve emails via a concept lattice following different paths. For the example above this means that one need not decide which of the two paths to use for storing. For retrieving the mail later, one can consider any combination of the catchwords[1] in the two paths.

Concept lattices are defined in the mathematical theory of Formal Concept Analysis [12]. A concept lattice is derived from a binary relation which assigns attributes to objects. In our application, the objects will be all emails stored by the system, and the attributes will be catchwords like 'conferences', 'mineau', and 'organisation'. We assume the reader to be familiar with the basic notions of

[1] By catchwords we mean small natural language phrases under which the user may meaningfully classify documents.

Formal Concept Analysis, and refer otherwise to [3] and to proceedings of past ICCS conferences.

There are related approaches to the above stated problem. For instance the concept of a *virtual folder* was introduced in a program called View Mail (VM) [6]. A virtual folder is simply a collection of email documents retrieved in response to a query. The virtual folder concept has more recently been popularized by a number of open source projects, e. g. [8]. Our system differs from those projects in both the understanding of the underlying structure via formal concept analysis, and the implementation.

Our approach is also related to the library information system implemented in the Center of Interdisciplinary Studies at Darmstadt University of Technology [7]. That system is based on the management system TOSCANA for Conceptual Information Systems [11]. The retrieval component of both our system and the library system provide basically the same functionality. The difference lies in the support for the user maintaining and updating the email collection. This is due to the fact that, while in the library system maintenance is allowed only to the librarian and/or a knowledge engineer, in an email management system storing emails is an essential and often used feature which requires some semi-automatic support for an untrained user.

In the next section, we will describe the mathematical structures of the Conceptual Email Manager. Requirements for their maintenance are discussed in Section 3. Issues related to an implementation of the requirements are discussed in Section 4. The paper is concluded by an outlook on future work.

In this paper we endeavor to precisely define the behavior of a natural user interface for managing emails based on Formal Concept Analysis. Although designing the interface to exhibit simple and rational behavior to the user, the exact semantics with respect to the underlying program structures the reader will find are rather detailed.

2 Structures Underlying CEM

We assume that the reader is familiar with the following two basic notions of Formal Concept Analysis: formal context and concept lattice. Definitions and examples can be found in [3] or in previous ICCS proceedings.

In this section, we describe the system on a structural level; we abstract from implementation details. They will be discussed in Section 3. Basically, we can distinguish three fundamental structures:

1. A *formal context* which assigns to each email a set of catchwords;
2. a *hierarchy* on the set of catchwords in order to define an information ordering over the catchwords;
3. and a mechanism for creating *conceptual scales* which are used within a graphical interface for the retrieval of emails.

These three structures are discussed in detail in the remainder of this section.

2.1 Assigning catchwords to emails

In the conceptual email manager, we use a *formal context* (G, M, I) for storing the emails and for assigning catchwords to them. The set G contains all emails stored in the system, the set M contains all catchwords. For the moment we consider M to be unstructured. (In the next subsection however, we will introduce a hierarchy on it.)

The relation I indicates which emails are assigned to which catchwords. In the example given in the introduction, the user might want to assign all the catchwords 'mineau', 'iccs2000', 'program_commitee', 'conferences', 'iccs', and 'organisation' to the new email. The incidence relation is generated in a semi-automatic process: (i) an automatic string-search algorithm may recognize words within sections of an email and suggest relations between the email and some attributes, (ii) the user may accept the suggestion or modify it, and (iii) she also may attach user defined attributes to the email. In Section 3, we will discuss how the user is supported in this assignment process. At the moment, we suppose that the relation is already given.

Instead of a tree of disjoint folders and sub-folders, we consider the concept lattice $\underline{\mathfrak{B}}(G, M, I)$ as navigation space. The formal concepts replace the folders. In particular, this means that emails can appear in different concepts. The most general concept contains all emails. The deeper the user gets in the hierarchy, the more specific are the concepts, i. e., the smaller is the number of emails they contain. Even so the user may, using general catchwords only, still obtain a great search depth from the conjunctions present in the concept lattice.

2.2 A hierarchy on the catchwords

In order to support the semi-automatic assignment of catchwords to the emails, we additionally provide the set M of catchwords with a partial order $\leq$. For this *subsumption hierarchy*, we assume that the following *compatibility condition* holds:

$$\forall g \in G,\ m, n \in M\colon \quad (g, m) \in I,\ m \leq n \ \Rightarrow\ (g, n) \in I \tag{‡}$$

(i. e., the assignment of catchwords to emails respects the hierarchy on the catchwords). Hence for assigning catchwords to emails, it is sufficient to assign the most specific catchwords only. All more general catchwords will be added automatically by the system. The maintenance of the hierarchy will be discussed in the two following sections.

As an example, the user may want to say that 'iccs' is a more specific catchword than 'conferences', and that 'iccs2000' is more specific than 'iccs' (i. e., 'iccs2000'$\leq$'iccs'$\leq$'conferences'). Emails regarding the production of this paper are then assigned by the authors to the catchword 'iccs2000' only (and maybe additionally to catchwords like 'cole' or 'stumme', and to 'papers'). When the authors want to retrieve these emails, they do not need to remember that they stored them under 'iccs2000'. They will also find them under the more general catchword 'conferences'. If this catchword provides a list of emails that is too

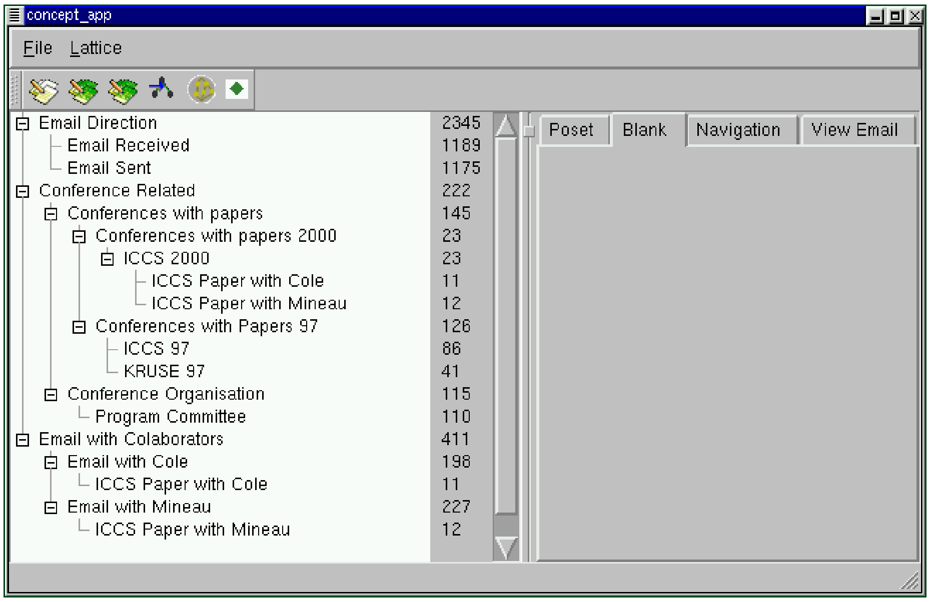

Fig. 1. Part of a catchword hierarchy

long, then they can either refine the search by taking a sub-term like 'iccs' or alternatively by adding another catchword, for instance 'cole'. The next subsection describes the structures which support the user in this kind of navigation.

While we note that it is not required by the theory that a particular structure be imposed on the hierarchy it is likely that the user will impose some structural notions on $(M, \leq)$. One appealing and natural notion is to split the hierarchy into three parts: One part related to contents of the emails, e. g., if an email is related to a conference or not, if it is used for its organization, etc. A second part related to the sender or receiver of the email. And a third part describing aspects of the mailing process (whether it is an inbound or an outbound mail etc.). An example of a hierarchy is given in Figure 1. (The right window of the screenshot is explained in Section 4.)

Even when the hierarchy imposed on the catchwords by the user is a tree, the resulting concept lattice — which we use as the search space — is by no means a forest. Consider for example the concept generated by the conjunction of the two catchwords 'ICCS 2000' and 'conference organization'. It will have at least two incomparable super-concepts, namely the one generated by the catchword 'ICCS 2000' and the one generated by the catchword 'conference organization'. In general, all we know is that the resulting concept lattice is embedded as a join-semilattice in the lattice of all order ideals of $(M, \leq)$ (i. e., all subsets X of M s. t. $x \in X$ and $x \leq y$ imply $y \in X$). [2]

[2] The use of this structure in the framework of knowledge discovery in databases is analyzed in more detail under the name of *power scale* in [5]. Refer also to the theorem of Birkhoff (stated for instance in [3, Theorem 39]).

2.3 Conceptual scales for navigating through the set of emails

Conceptual scaling has been introduced in order to deal with many-valued attributes. Often attributes are not one-valued, as for instance with the catchwords given above, but instead allow a range of values. This is modeled by a *many-valued context.* A many-valued context is roughly equivalent to a relation of a relational database with one field being a primary key. As one-valued contexts are special cases of many-valued contexts, conceptual scaling can also be applied to one-valued contexts in order to reduce the complexity of the visualization.

In this paper, we only deal with one-valued formal contexts. Readers who are interested in the exact definition of many-valued contexts and the use of conceptual scaling in this more general case are referred to [3]. Applied to one-valued contexts, conceptual scales are used to determine the concept lattice which arises from one vertical 'slice' of a large context:

Definition 1. *A* conceptual scale *for a subset $B \subseteq M$ of attributes is a (one-valued) formal context $\mathbb{S}_B := (G_B, B, \ni)$ with $G_B \subseteq \mathfrak{P}(B)$. The scale is called* consistent *with respect to $\mathbb{K} := (G, M, I)$ if $\{g\}' \cap B \in G_B$ for each $g \in G$. For a consistent scale $\mathbb{S}_B$, the context $\mathbb{S}_B(\mathbb{K}) := (G, B, I \cap (G \times B))$ is called its* realized scale.

Conceptual scales are used to group together related attributes. They are determined as required by the user, and the realized scales are derived from them when a diagram is requested by the user.

The Conceptual Email Manager stores all scales which the user has defined in previous sessions. To each scale, she can assign a unique name. This is modeled by a mapping.

Definition 2. *Let $\mathcal{S}$ be a set, whose elements are called* scale names. *The mapping*

$$\alpha\colon \mathcal{S} \to \mathfrak{P}(\mathcal{M})$$

defines for each scale name $s \in \mathcal{S}$ a scale $\mathbb{S}_s := \mathbb{S}_{\alpha(s)}$.

For instance, the user may introduce a new scale which classifies the emails according to being related to a conference by adding a new element 'Conference' to $\mathcal{S}$ and by defining $\alpha(\text{Conference}) := \{\text{CKP '96}, \text{AA 55}, \text{KLI '98}, \text{Wissen '99}, \text{ICCS 2000}\}$.

Observe that $\mathcal{S}$ and M need not be disjoint. This allows for instance the following construction which deduces conceptual scales directly from the subsumption hierarchy: Let $\mathcal{S} := \{m \in M \mid \exists n \in M\colon n < m\}$, and define, for $s \in \mathcal{S}$, $\alpha(s) := \{m \in M | m \prec s\}$ (with $x \prec y$ if and only if $x < y$ and there is no z s.t. $x < z < y$). This means that all catchwords $m \in M$ which are neither minimal nor maximal in the hierarchy are at the same time considered as the name of scale $\mathbb{S}_m$ and as catchword of another scale $\mathbb{S}_n$ (where $m \prec n$). In this paper, we will call scales constructed this way *default scales.*

This last construction has first been presented in [10] for defining a hierarchy of conceptual scales for the library information system [7]. In [10], however, only this special construction was considered. It turns out that, in general, a more flexible construction is desirable. In the library information system, for instance, one is also interested in scales for the minimal elements in $(M, \leq)$. Each such scale $\mathbb{S}_m$ has as attributes the upper covers of m (i.e., all $n \in M$ with $m \prec n$). This construction is made possible by using the function α which we have introduced in this paper.

3 Requirements of the Conceptual Email Manager

In this section, we discuss requirements of a conceptual email manager based on the paradigm of Formal Concept Analysis. In the following section we shall explain how our implementation responds to these requirements.

The requirements may be divided along the same lines as the underlying mathematical structures defined in Section 2. Briefly stated the requirements are:

1. to assist the user in building, browsing and modifying the catchword hierarchy;
2. to help the user modify the scale function α;
3. to allow the user to manage the assignment of catchwords to email documents; and
4. to assist the user in searching the conceptual space of emails for both individual emails, and also conceptual groupings of emails.

In addition to the requirements stated above, a good email system needs to be able to send, receive and display emails by processing the various email formats and interacting with the current popular email protocols. Since these requirements are already well understood and implemented by existing email programs they will not be discussed further in detail in this paper.

Browsing and Modifying the Catchword Hierarchy. The catchword hierarchy is a partially ordered set $(M, \leq)$ where each element of M is a catchword. Listed below are requirements related to browsing and modifying of the catchword hierarchy.

1. The program should display graphically the structure of the partial order $(M, \leq)$. The ordering relation must be clearly evident to the user.
2. It must be possible, via a series of graphical manipulations initiated by the user and implemented in the program to add and to delete elements and to alter the ordering relation. It should be possible to create any partial order within a reasonable size limit.

Modifying the Scale Function. The user must be able to modify the scale function α, explained in Section 2. Therefore the tool should provide a suitable visualization of the function. The program must allow an overlap between the set $\mathcal{S}$ of scale names, and the set M of catchwords.

Managing the Assignment of Catchwords to Emails. The program should store the formal context (G, M, I) and ensure that the compatability condition (‡) is always satisfied. It is inevitable that the program will have sometimes to modify the formal context, after a change is made to the catchword hierarchy, in order to satisfy the compatability condition. This modification can be made either automatically, or via an interactive process where the user is asked whether the changes should be made.

The program must support two mechanisms for the association of catchwords to emails. Firstly there should be a mechanism as described in Section 2.1 by which emails are semi-automatically associated with catchwords based on the email content. Secondly the user should be able to view and modify the association of catchwords with emails.

Navigating the Conceptual Space. The program should assist the navigation of the conceptual space of the emails by drawing line diagrams of concept lattices arising from conceptual scales [3]. These line diagrams should extend to locally nested line diagrams [9, 10]. The program must allow the retrieval and viewing of emails that form the extension of concepts displayed in these line diagrams.

4 Implementation

This section divides the description of the implementation of our conceptual email manager, CEM, into a structure similar to that presented in Section 3.

4.1 Catchword Hierarchy

Browsing the Hierarchy. The user is presented with a view of the hierarchy, $(M, \leq)$ as a tree widget,[3] shown in Figure 1. The tree widget has the advantage that most computer users are familiar with its operation, and that it provides a compact representation (in the sense of space used on the screen) of a tree structure.

The catchword hierarchy, being a partially ordered set, has a more general structure than that of a tree. No limitation is placed by the program on the structure of the partial order in general. Following is a definition of the tree derived from the catchword hierarchy with the purpose of defining the contents and structure of the tree widget.

Let $(M, \leq)$ be a partially ordered set and denote the set of all sequences of elements from M by M^* (including the empty sequence ε). Then the labeled tree derived from the catchword hierarchy is comprised by $(T, \sqsubseteq, \texttt{label})$ where $T := \{(m_1, \ldots, m_n) \in M^* \mid m_i \prec m_{i+1},\ m_n \in \max(M)\} \cup \{\varepsilon\}$, $w_1 \sqsubseteq w_2$ iff w_2 is a suffix of w_1, and $\texttt{label}\colon T \setminus \{\varepsilon\} \to M$ is the function defined by $\texttt{label}(m_1, \ldots, m_n) := m_n$.

[3] A widget is a graphical user interface component with a well defined behaviour usually mimicking some physical object.

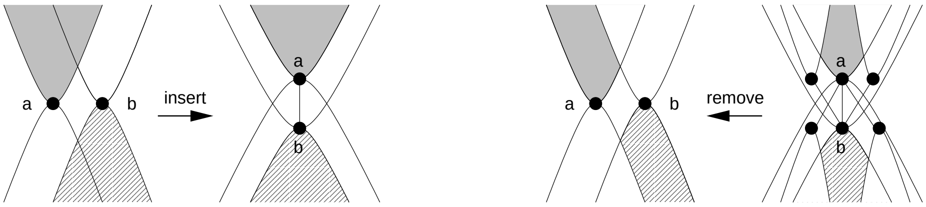

Fig. 2. `Insert` and `removal ordering` operation

Each tree node is identified by a path from a catchword to the top of the catchword hierarchy. Although the tree representation has the disadvantages that elements from the partial order occur multiple times in the tree and that the tree can become large, the saving of space and the regular structure are our reasons to prefer it to other order representations. If the user keeps the number of elements with multiple parents in the partial order to a small number then the tree is manageable.

Modifying the hierarchy $(M, \leq)$. The program provides four operations for modifying the hierarchy: `insert catchword`, `remove catchword`, `insert ordering` and `remove ordering`. More complex operations provided to the user, for example moving an item in the taxonomy, are resolved internally to sequences of these four operations. In this section we denote the order filter (also called the up-set) of m as $\uparrow m := \{x \in M \mid m \leq x\}$, the order ideal (also called the down-set) of m as $\downarrow m := \{x \in M \mid x \leq m\}$, the lower cover of m as $\prec_m := \{x \in M \mid x \prec m\}$, and the upper cover of m as $\succ_m := \{x \in M \mid x \succ m\}$.

The operation `insert catchword` simply adds a new catchword to M, and leaves the $\leq$ relation unchanged. This means that the new catchword is incomparable to all other catchwords. The `remove catchword` operation takes a single parameter $a \in M$, and simply removes a from M and $((\downarrow a) \times \{a\}) \cup (\{a\} \times (\uparrow a))$ from the ordering relation.

The operation `insert ordering` takes two parameters $a, b \in M$ and inserts into the relation $\leq$, the set $(\downarrow b) \times (\uparrow a)$. The operation has been drawn in the left diagram in Figure 2 which serves as a form of Venn-Diagram for the up-sets and down-sets of a and b before and after the insert operation. The shading gives an indication of corresponding regions.

The insertion of the ordering $b \leq a$ into $\leq$ will require the insertion of the set $\{g \in G \mid (g, b) \in I\} \times (\uparrow a \setminus \uparrow b)$ into I. The portion of M whose image under the relation I will require an update is the upper shaded part in the rightmost diagram in Figure 2.

The operation `remove ordering` takes two parameters $a, b \in M$ where a is an upper cover of b. The `remove ordering` operation removes from $\leq$ the set $((\downarrow b) \setminus (\downarrow(\prec_a \setminus \{b\}))) \times ((\uparrow a) \setminus \uparrow(\succ_b \setminus \{a\}))$. The right diagram in Figure 2 may be used to visualize the remove operation. Similarly to the insert operation, the removal of the ordering $b \leq a$ from $\leq$ will require a re-computation of the image in I under the elements from $\{a\} \times ((\uparrow a) \setminus \uparrow(\succ_b \setminus \{a\}))$. This region has been shaded in the upper right of Figure 2.

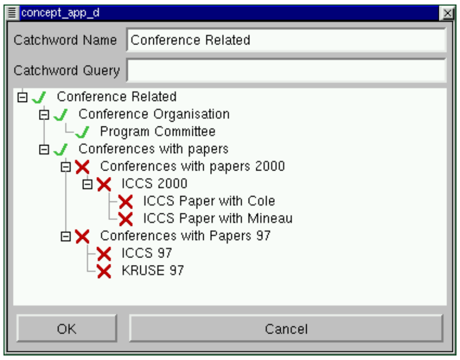

Fig. 3. Dialog for editing α(Emails with Cole)

4.2 Modifying the Scale Function

The set $\mathcal{S}$ of scale names, as explained in Section 2, is not necessarily disjoint from M, thus the tree representation of M already presents a view of a portion of $\mathcal{S}$. In order to reduce the complexity of the graphical user interface, we make $\mathcal{S}$ equal to M. That is: all catchwords are scale names, and all scale names are catchwords. Such an assumption is made possible by the definition, given in Section 2, of the default scale for a catchword. A result of this definition is that catchwords with no lower covers will map, under the scale function, α, to the empty set.

The function α maps each catchword m to a set of catchwords. The program displays this set of catchwords, when requested by the user, using a dialog (see Figure 3). The dialog box contains a set of catchwords available for membership in $\alpha(m)$. In Figure 3 this set of candidates has been restricted to the down-set of m. An icon (either a green tick or a red cross) is used to indicate membership (or not) in the set of catchwords given by $\alpha(m)$. By clicking on the icon the user can change the definition of $\alpha(m)$.

By displaying only the down-set of m in the dialog box, the program restricts the definition of α to $\alpha(m) \subseteq (\downarrow m)$. This restriction has an effect on the "remove ordering operation" defined on $(M, \leq)$. When the ordering of $a \leq b$ is removed the image of the function α for attributes in $\uparrow a$ is automatically checked and if necessary modified.

The program has an intended mode of operation for expert users in which the restriction on the definition of $\alpha(m) \subseteq \downarrow m$ is lifted. In this mode the user has all catchwords available for inclusion in $\alpha(m)$, and he may choose the set $\mathcal{S}$ of scale names to be different from the set M of catchwords.

When the function α is changed by the user then the set $\{\mathbb{S}_s \mid s \in \mathcal{S}\}$ of scales is changed automatically. This update occurs regardless of the mode of operation. The new/modified scales can then be used directly for navigating in the concept space as described in Section 4.4.

4.3 Associating Emails with Catchwords

Each member of $(M, \leq)$ is associated with a query term, which in this application is a set of *section/word pairs*. For our purposes a *section* of an email is either a header field, e.g. the "From:" field, or the section "body" which is composed of the parts[4] of the email directly encoding text. More formally stated: Let H be the set of sections found in the email documents, W the set of words found in the email documents, then the function $\texttt{query}\colon M \to \mathfrak{P}(H \times W)$ attaches to each attribute a set of section/word pairs.

Let G be a set of email documents. Five relations, Q, R, R^+, R^-, and I are defined for managing the different ways in which email documents may be associated with catchwords. $Q \subseteq G \times (H \times W)$ is a relation between documents and section/word pairs. The relation member $(g, (h, w)) \in Q$ indicates that document g has word w in section h. Q is stored via an inverted file index and is only updated when new email is presented to the system. The relation $R \subseteq G \times M$ is derived from the relation Q and the function $\texttt{query}$ via: $(g, m) \in R$ iff $(g, (h, w)) \in Q$ for some $(h, w) \in \texttt{query}(m)$. The relation R is only used as an intermediate step and is calculated from Q as required by the program.

The relations R^+ and R^- store user judgments saying whether or not an email should have a catchword m. These judgments will "over-rule" the relation R. We impose the constraint

$$((\uparrow R^+) \cap (\downarrow R^-)) \stackrel{!}{=} \emptyset \quad (\#)$$

on the two relations R^+ and R^-, saying that a user is not allowed to contradict himself. I. e., he is not allowed, for $m \geq n$, to assign (g, m) to R^- and (g, n) to R^+.

The relation I respecting the compatibility condition ($\ddagger$) is derived from the relations R, R^+ and R^- using the following operator: For any relation $J \subseteq G \times M$, we define $J^\ddagger := \{(g, m) \in G \times M \mid \exists n \in M\colon (g, n) \in J,\ n \leq m\}$. We obtain I as the incidence relation for the formal context (G, M, I) mentioned in Section 2 by $I := ((R \setminus R^-) \cup R^+)^\ddagger$.

These five relations are required to accommodate the different ways in which an email may be associated with catchwords. Q and R associate emails with catchwords via an automatic process based on content and queries attached to catchwords, R^+ and R^- associate email based on user input, and I combines these two sources with the hierarchy defined over the catchwords. By separating the relations for automatic associations of catchwords to emails from the relations for user defined associations, the program maintains a pure keyword index into the email collection. Relations R and I are derived from Q, R^+, and R^-, and so need not be stored. Storing I however greatly reduces the time complexity of the program.

When a batch of new emails, G_b, is presented to the program, the relation Q is updated automatically by inserting new pairs, Q_b, into the relation. The

[4] The MIME extension to the email format allows an email document to have multiple parts. These multiple parts are sometimes referred to as attachments.

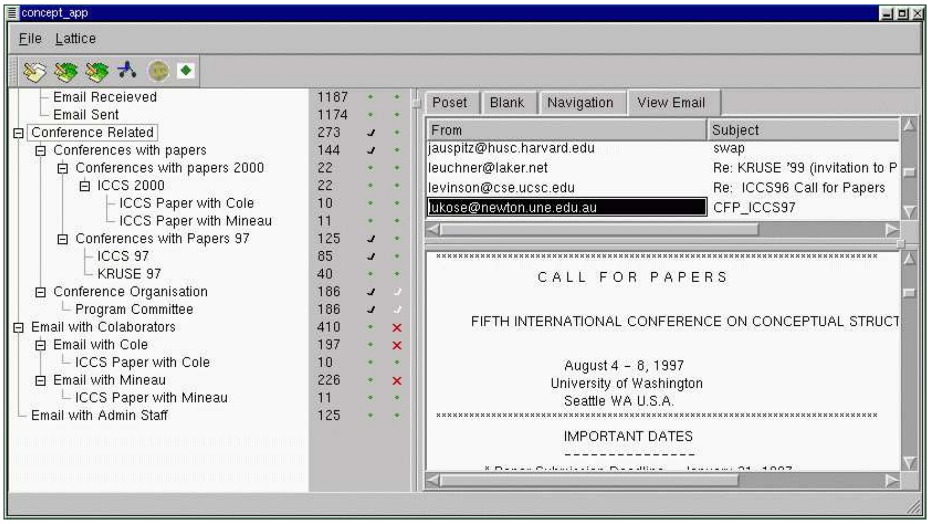

Fig. 4. Interface for viewing email and associating email with catchwords

modification of Q into $Q \cup Q_b$ will cause an insertion of pairs R_b into R according to `query`(m) and then subsequently an insertion of new pairs I_b into I. The definitions are:

$$Q_b \subseteq G_b \times (H \times W)$$
$$R_b = \{(g,m) \mid \exists\,(h,w) \in \text{query}(m) \text{ and } (g,(h,w)) \in Q_b\}$$
$$I_b = \{(g,m) \mid \exists\, m_1 \leq m \text{ with } (g,m_1) \in R_b\}$$

The user can modify the association of emails with catchwords in two ways. Firstly by changing the relations R^+ and R^- and secondly by making modifications to the `query` function. In order to explain the user interface for making modifications to R^+ and R$^-$ we introduce the following notation. For an email $g \in G$, we define the restriction of any relation $J \subseteq G \times M$ to this email by $J_g := J \cap (\{g\} \times M)$. For the purpose of brevity of expression we shall say m belongs to J_g if $(g,m) \in J_g$.

The user is able to view individual emails as shown in Figure 4. In this mode icons are attached to catchwords in the tree widget displayed to the left of the email. These icons indicate to the user how each of the catchwords is related to the displayed email by R, R^-, and R^+. The user is able to change the relations R^- and R^+ by interacting with the icons.

1. If m is not in $R_g^\ddagger$, $(R_g^+)^\ddagger$, or R_g^- then no icon is displayed.
2. If m is in $R_g^\ddagger$ then a yellow tick (shown as white in Fig. 4) is displayed.
3. If m is in R_g^- then a red cross is displayed.
4. If m is in $(R_g^+)^\ddagger$ then a green (shown as black in Fig. 4) tick is displayed.

All combinations of these icons which do not include at the same time a red cross and a green tick are possible.

The user can then determine that the displayed email has a catchword in I if there is either a green tick or a yellow tick in the absence of a red cross. The program provides two basic operations, `associate attribute` and `disassociate attribute` from which more complex operations for use in the user interface may be constructed. The `associate attribute` operation takes two parameters, an email document, and a catchword m. The operation inserts the pair (g, m) into R^+, and removes, for all $n \geq m$, (g, s) from R^-. Similarly the operation `disassociate attribute` takes two parameters, an email and a catchword. The operation inserts (g, m) in R^- and removes, for all $n \leq m$, (g, n) from R^+. The construction of the two operators guarantees that the constraint (#) is always satisfied.

The user is also able to influence the way that R is derived from Q by modifying the `query` function. The user is able to modify $\alpha(m)$ using the Scale Query field in the dialog box shown in Figure 3. After any such modification to the `query` function the relations R and I are modified accordingly.

New emails presented to the system for automated indexing cause a modification to the inverted file index consisting only of new entries. The insertion of new email documents into an inverted file index is an efficient operation. The complexity of inserting each document is $O(1)$. When the user makes a modification to either R^+ or R^- of a removal or insertion of (g, m) this will cause all catchwords in the order filter of m, or order ideal, resp., to be updated in I. The expense of such an update depends on how I is stored but is likely to be $O(log(n))$ where n is the average number of documents per attribute.

It is useful for the system to maintain the relation R^+ for special catchwords dependent on observation by the program of the users behavior. Two examples of such catchwords are "read emails" for emails that the user has displayed at some time, and "unseen emails" for emails that the user has not yet been notified of.

4.4 Navigating the Conceptual Email Space

To assist the user in navigating the conceptual space of emails, the program draws simple line diagrams and (locally) nested line diagrams. A simple line diagram is used to visualize a single scale, while nested line diagrams are used to visualize combinations of scales. The concept lattices, from which the nested line diagrams are drawn, are computed from the contexts given by $\mathbb{S}_{\alpha(s)}$. The contexts are calculated using the algorithm reported in [1], and the concept lattices are calculated from these contexts via Ganter's algorithm [3].

The user may navigate the conceptual space of emails documents for different purposes:

1. to find collections of emails thematically linked;
2. to review the precision and recall of queries attached to catchwords by comparing them with catchwords based on user judgments (for the purpose of refining them for improving the `query` function); and
3. to review patterns of communication between different groups.

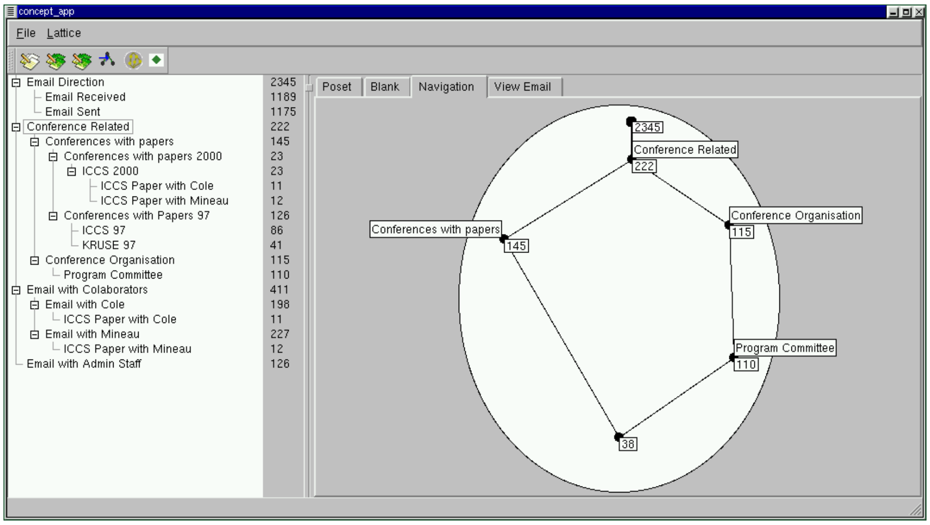

Fig. 5. Concept Lattice derived from the Scale for "Conference Related".

Of these purposes the first is the most useful to common users of the program. A simple scenario in which the user has this first purpose is presented here. Imagine a researcher who was in the Program Committee (PC) of ICCS '97 and was at that time co-authoring with other members of the PC for the same conference. For the organization of a conference in the year 2000, she wants to retrieve some facts about the organization of ICCS '97. But she only remembers that she exchanged this information with one of the people she was co-authoring with for ICCS '97, and that it was only one tiny part of a mail covering all kinds of topics.

The researcher may begin her search by requesting a line diagram for the scale named "Conference Related". This scale is shown in Figure 5. It shows that from her 2344 emails in total, there are 222 emails related to conferences, 145 of which are related to conferences with papers submitted and 110 of which are related to both conference organisation and program committees. The researcher decides that the email she is looking for is likely to be under the catchword "Conferences with Papers". As there are too many emails in its extent to be read through, she may for instance want to expand the concept. By choosing the scale $\mathbb{S}_{\text{Conferences 1997}}$, she obtains Figure 6.

Now the researcher can for instance check the 19 mails related to "ICCS '97" and "Conference Organization/Program Committee". If she still doesn't find the email she is looking for there, then she has to check either the 86 papers related to "ICCS '97" or even all 115 emails under the catchword "Conference Organization". Before doing this, however, she might want to differentiate these concepts further, e. g. by zooming into them with the scale "Members of ICCS '97 Program Committee". If this scale doesn't exist yet, then she can create it on

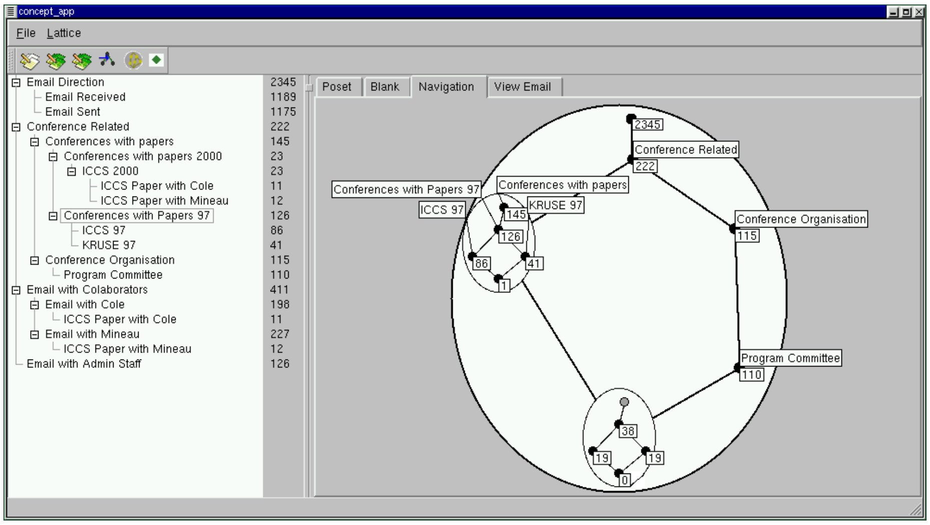

Fig. 6. Concept Lattice derived from the Scales for “Conference Related” and “Conferences with Papers”.

the fly using the widget for modifying the scale function (and eventually store it for further use).

Note that with a classical, tree-structured search hierarchy (where one usually has the names of the correspondents on the highest level), one would be forced to scan all branches starting with the names of the co-authors before one can tell the system constraints like “Conference Related”.

5 Outlook

Having completed a prototype implementation of CEM (available on request from the first author, the next step is to evaluate its operation in daily use and to measure is scalability with respect to large data sets and distributed collections of email. We also consider allowing the user to impose more structure on M including conjunctive implications and negation, following the mathematical foundation presented in [4], as well as a more expressive language for the `query` function which allows for instance disjunctive queries.

Although CEM has been in this paper applied to email documents it has a more general use as a document management system. The next step therefore is to extend the current architecture to allow the user to associate catchwords with files accessible remotely via the internet and also locally with the users private collection. This next step has the challenge of dealing with a large number of protocols and file formats. The emergence of standards such as XML and RDF gives some hope for a general and unified method for processing of this myriad of data formats. Looking further ahead one can consider how a conceptual file management system might be used in a group environment or at an enterprise level where several users contribute to the structure of the hierarchy and the association of catchwords with files.

References

1. R. Cole, P. Eklund: Scalability in formal concept analysis: a case study using medical texts. *Computational Intelligence* Vol. 15, Number 1, 1999, 11–27
2. A. Fall: Dynamic taxonomical encoding using sparce terms. *Proc. ICCS '96*. LNAI **1115**, Springer, Heidelberg 1996, 278–292.
3. B. Ganter, R. Wille: *Formal Concept Analysis: mathematical foundations.* Springer, Heidelberg 1999 (Translation of: Formale Begriffsanalyse: Mathematische Grundlagen. Springer, Heidelberg 1996)
4. B. Ganter, R. Wille: Contextual Attribute Logic *Proc. ICCS '99*. LNAI **1115**, Springer, Heidelberg 1999, 377–388.
5. J. Hereth: *Formale Begriffsanalyse im Data Warehousing.* Diploma thesis, TU Darmstadt 2000
6. K. Jones: View Mail Users Manual. *http://www.wonderworks.com/vm.* 1999
7. T. Rock, R. Wille: Ein TOSCANA-System zur Literatursuche. In: G. Stumme and R. Wille (eds.): *Begriffliche Wissensverarbeitung: Methoden und Anwendungen.* Springer, Berlin-Heidelberg 2000, 239–253
8. W. Schuller: *http://gmail.linuxpower.org/.* 1999
9. G. Stumme: Local Scaling in Conceptual Data Systems. *Proc. ICCS '96*. LNAI **1115**, Springer, Heidelberg 1996, 308–320.
10. G. Stumme: Hierarchies of Conceptual Scales. *Proc. Workshop on Knowledge Acquisition, Modeling and Management.* Banff, 16.–22. Oktober 1999, Vol. 2, 5.5.1–18
11. F. Vogt, R. Wille: TOSCANA — A graphical tool for analyzing and exploring data. In: R. Tamassia, I. G. Tollis (eds.): *Graph Drawing '94*, LNCS **894**, Springer, Heidelberg 1995, 226–233
12. R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.): *Ordered sets.* Reidel, Dordrecht–Boston 1982, 445–470