On Querying and Exploring Activities on a User's Desktop

Sibel Adalı & Shawn Pearce
Department of Computer Science
Rensselaer Polytechnic Institute
Email: {sibel,pearcs}@cs.rpi.edu

Maria-Luisa Sapino
Department of Computer Science
Università di Torino
Email: mlsapino@di.unito.it

***Abstract*— Many desktop query and management systems offer an object oriented view of a computer, where the emphasis is on describing the information relevant to an object. However, objects may be related to each other in different ways and in different contexts. In this paper, we argue that users create and modify data as a function of activities that they are involved in. We develop methods to explore objects on a desktop through activities. Our methods integrate discovered relationships between data objects based on their participation in different activities as well as other properties. Our activity model is well suited to support a query language that is able to alter the context and the definition of an activity to easily visualize complex relationships in data. We show that this new organization makes many new interesting desktop functionalities a reality.**

I. Introduction

In this paper, we introduce activities as a new mechanism for grouping related objects on a person's desktop. Data are created and manipulated every day in an ever increasing rate. To find and organize data, we need to develop methods to digest it easily. Methods are now being developed to extract relationships [8] between data objects based on their properties and to attach new relationships to data objects with the help data mining methods [1], [15] and user interfaces [12], [17]. In this paper, we propose to enhance these methods with the notion of an activity, which can be loosely defined as a view definition that describes which objects are related to an activity. We show that such a grouping makes it possible to develop novel query and data organization methods for a user's desktop.

Similar to previous work in this area [12], we aim to develop a user-centric view of the world which aims to organize data based on the interests and goals of a specific user. A person normally creates data in many different ways, some on her own computer(s) and some through her participation in group activities such as maintaining wikis, sharing photos and developing code. To complement this, people either known or unknown to the user create data that the user accesses or receives. These outside events are sometimes caused by the user (such as a reply to a previous email) or are simply triggered externally (such as receiving an email when the school is closing due to a snow storm). In this paper, we are interested in organizing data that is of interest to the current user. We will concentrate on data that is on one's own desktop, but our methods can easily be extended to any data outlet a user participates in. The complementary problem of organizing events [19] is being studied by other research groups, as activities originate from a user, we view events as concepts external to the user, such as the war breaking. The interplay between the two concepts is an interesting research problem in itself.

As shown in [8], organizing data around known properties of objects allows users to search for data based on multiple properties beyond simple text search and show other objects sharing the properties of a given object. This allows users to find objects based on their relationship to each other. However, objects may be related to each other even if they do not share common properties. To this end, Haystack [12] proposes activities which are tags that can be assigned to single or groups of objects. Similar concept of smart folders in Mac OS.X operating system uses a boolean condition to group related objects. Given an example of an activity, then it is possible to use data mining methods [15] to find similar objects. All of these methods consider the objects as the main data objects are searched and grouped by their properties. While these methods are definitely a very useful approach to organizing data, they do not address the issue of scale properly. It is already quite hard to tag individual objects or even organize them in data folders for some users. Defining complex boolean conditions is then a much bigger challenge for such users. To complicate the problem further, what is of interest to a user may change quickly. Hence maintaining the tags, example objects and definitions can easily become very time consuming operations. While searching for objects by properties is an absolutely necessary first step, it does not bring together all related objects. Furthermore, we might not only be interested in grouping related objects, but we might also want to observe how these relationships change over time. We might want to limit our searches to some objects or remove certain objects from consideration. For example, a search at work should not return objects related to our personal interests. A search system should allow us to change the focus of our queries to specific contexts easily. For example, we might want find the most relevant objects for a specific time frame based on data that was available at that time. What is relevant to a query may differ greatly from one day to the next.

In this paper, we introduce a new framework that extends the previous work in this area and introduces a novel framework

for querying and exploration of data based on the concept of activities. We make three unique contributions. First, we define an activity as a general concept for grouping semantically related objects and show activities can be defined with view definitions that reflect users' interests and preferences. One of the main problems for any such approach is that it is unreasonable to expect naïve users to define complex activities that serve their purpose. With our second contribution, we show that it is possible to extract activities from data objects automatically. We illustrate one such method for extracting activities and provide some results. We show that by organizing data around the notion of activities, we are able to get more accurate results then the usual object based approaches. Finally, we show how these two approaches can be combined to implement interesting functionalities that go beyond keyword search. We illustrate five different methods to organize data that are being implemented as part of our activity browser interface. Our browser is being built as an extension of keyword based search interfaces (such as the Spotlight interface of Mac OS.X) and association based browsers (such as Haystack [12]).

II. Related Work

One of the goals of information management applications is to take into account users' needs and requirements when making information available to them. In particular, it is important for these systems to offer data management services and interfaces that can easily adapt to users' skill and experience.

When the available information is stored on the users' desktops, it is extremely important for such systems to be able to model users' interpretation of their data and to capture the possibly different meanings, semantics links, and relationships that the users associate to the information units available. For this purpose, various Personal Information Management tools are being developed to assist the user with her navigation/browsing over various forms of personal digital data [6], [9], [8], [12], [17], [20].

MyLifeBits [17] is a research project and a software environment which aims at storing, in digital form, *everything* related to the activities of an individual and providing full-text search, text and media annotations, and hyperlinks to personal data. Another Microsoft project, *Stuff I've Seen* [9], aims at managing personal data, such as already-read email messages, for reuse. Retrieval and presentation of information are based on contextual cues, such as time and author in the case of email. Lifestreams [10] organizes user's data as a data stream similar to our approach and introduces operators to find and summarize data in a stream. But, there is no equivalent notion of activities in this work.

Recently, there is more work on personal desktop information management. Chandler [6], for instance, is an interesting open source example of such management tools, integrating calendar, email, contact management, task management, notes, and instant messaging functions. Haystack [12] and Gnowsis [20] are systems that adopt the semantic web data modeling approach, and treat all the data objects stored on the desktop as resources on which semantic networks are defined using the Web Consortium's *Resource Description Framework (RDF)* [18]. In Haystack, a number of parsers are available to be appropriately triggered when data extracted from traditionally formatted data (such as directory hierarchies, various formats of documents, music, e-mails, bibtex data, photographs) are to be incorporated into the system and need to be translated in RDF. The information storage and retrieval mechanism of Haystack relies on a structured database approach. *Related* objects are rendered to the user through a recursive process, in which each object is asked to render itself and recursively sends the same request to other objects that it is related to. The use of *lenses* allows the different properties that characterize a specific aspect of the information being presented to be grouped together. Data models describe appropriate lenses for different types of object. Haystack also makes it possible to mark objects with a specific tag which can be used to mark objects belonging to a specific activity, but these tags are not treated any differently than any other property that the objects may have and are crisp sets by definitions.

Another Semantic Web based approach to deal with desktop information, treats different applications as resources, is used in *Gnowsis* [20]. In this case, both resources' metadata and relationships between all the resources on a computer are represented using the Resource Description Framework (RDF) rules. Data from different applications are linked using RDF triples. The information is visualized by means of a Semantic Browser. One major difference of Gnowsis from Haystack is the explicit goal of integrating existing applications and tools instead of providing an entirely new environment. Thus, Gnowsis integration framework makes it possible to extract information on the fly from common desktop applications.

More user centered treatment of object semantics recently lead to a new emerging research area referred to as *Experiential Computing* [2], [4], [11]. According to this approach, the user interaction systems should exploit and reflect as closely as possible users' previous experience. Thus, users should be part of the complete system. Experiential environments allow a user to directly observe data and information of interest related to an event and to interact with the data based on his or her own interests in the context of that event. By developing experiential environments, researchers aim to develop new generation information management systems which transform database applications from being simply information sources to being powerful insight and experience sources. The data generated for each event is experienced by an observer and interpreted to create knowledge. In this knowledge production process, the observer plays an important role to interpret the data, and capture the experienced semantics.

The closest to our approach is the work by Dong and Halevy [8] and the IRIS [7] system that they contribute to. The work in [8] introduces a structured data model over the database of objects. But, this is mostly a relational view which is an essential first step for our approach also. The IRIS [7] framework introduces common sense reasoning to reason about related objects. To our knowledge, there is no

notion of activities similar to our definition here in this work. As in Haystack, they introduce a single system for storing and accessing desktop data. Our approach is to build the browser independent of other applications and use basic information about the objects to infer activities.

Our approach differs from all of the above systems in various ways. Most important one is the notion that objects in a desktop may be related to each other in different ways in different contexts. These context, we argue, can be modeled by means of activities that causes the users to create and modify data. The relatedness of an object to an activity is a fuzzy notion. We develop methods to define and query activities. This allows users to not only locate relevant information but also organize their desktop in relationship to these activities.

III. Activities

As described in the introduction, our aim in this paper is to define and retrieve objects relevant to an activity. In our model, an activity is a general concept that causes the creation of data. However, we do not want to model the meaning of the activity. We simply want to define which data objects are relevant to this activity for a specific user. For example, writing a paper, taking a class, going on a trip are examples of activities that have a temporal component. The objects related to the "paper writing" activity may range from the files and communication related to the paper, to the submission to a conference and the reservations and registration for the conference and our web queries on the city being visited on the trip. Some activities can be thought of as ongoing such as our background web surfing, the blog we maintain and the topics we discuss there. Regardless of the type of activity we are involved in, the activities cause the creation and modification of data objects. To this end, we will define activities based on how they relate to the data objects. First, we introduce some terminology that will be used in the following sections.

A. Basics

In this paper, we assume a simple object-oriented view of a computer. A data object is used to refer to any type of file stored on the computer. We assume the computer stores a set of objects DDB which is a subset of the universe $\mathcal{O}$ of all possible objects that can be stored on one's computer. At a specific time point t, the contents of the desktop is identified by DDB_t. Examples of data objects are text files, powerpoint presentations as well as different application files, images, video clips, edited movies which may contain images and video clips, e-mails, calendar entries, people identified in various ways, keywords used in desktop and Internet search queries, and URLs of visited web pages. As in the object-oriented database models, these objects have attributes and these attributes have values and associated methods. For example, mails have `to, from, cc` fields that all take values of type `person`. However, some attributes may have time or space varying values such as the update time and location for a file. We do not explicitly model these but assume the existence of appropriate methods for querying these. For example, for time the method `about(X)` may be implemented to check how close a data value is to the input date `X`.

To query the objects in a database DDB of objects, we use simple selection conditions using the attributes and methods over the objects. We also assume that we can retrieve all objects of a specific type as a collection where `ALL` refers to the union of all objects. Hence, we can write simple queries of the form:

```
SELECT DISTINCT O1
FROM ALL O1, ALL O2
WHERE O1.type = 'mail' and
O2.type='person' and O1.sender=O2 AND
O2.ANY = 'Jane Doe'
```

which returns all mail objects where the sender is the person `'Jane Doe'`. This hides some complexity in the processing of this query since mail sender is probably an email, which is mapped to a person known to the user. This person can also be identified by the text query `'Jane Doe'` that is searched on any attribute of this object. Reconciling references to the same object and finding that a person referred in two different ways is the same person is a research topic in data cleaning research [8], [14] which we do not directly address in this paper. However, we note that finding objects that are related, similar to each other is an important step in developing effective data cleaning methods, which is the focus of our paper.

For the remainder of the paper, we will only be interested in the above type of queries containing only simple select, project and join predicates that return objects of a specific type. We will use the shorthand

`Mail(sender` $\sim$ `Person(ANY` $\sim$ `'Jane Doe'))`

to refer to this condition where $\sim$ refers to the fuzzy evaluation condition which will be referred to as `contains`. We will call these selection conditions as defined below:

Definition 1: [Selection Condition] An atomic selection condition is given by an expression of the form `O(A` $\sim$ `c)` where `O` is any object type or `ALL`, `c` is a simple value (i.e. a primitive value like a string or an integer or an object), and `A` is an attribute of objects of type `O` or the keyword `ANY` or a method that maps objects of type `O` and value `c` to a score in $[0, 1]$.

A selection condition is either an atomic selection condition or an expression of the form `O(A` $\sim$ `c)` where `O` and `A` are the same as above, but `c` is a selection condition.

Given this terminology, we can define new selection conditions as given below:

```
p_1= Mail(sender ~
        Person(email ~ jd@somewhere.com))
p_2= Mail(sentDate~ Date(6/8/2005))
p_3= Directory(name ~ 'photos/2005')
p_4= Site(URL ~ 'www.cs.rpi.edu')
p_5= File(keyword ~ 'plane ticket')
```

are examples of conditions specific using properties that objects may have. These properties correspond to mails sent by a specific person or sent at a specific date, files in a directory

with a specific name, or URLs visited from a given site. The last property refers to files with the keywords `plane, ticket`.

Given that some selection conditions return a list of objects with a score of match, we need to define the satisfaction of these conditions. For example, the keyword query above would normally assign objects containing both keywords a higher rank than those that only contain one of the keywords.

Definition 2: [Satisfaction of selection conditions] Suppose, we are given a database DDB of objects and combination functions f_{AND}, f_{OR} for the appropriate connectives. A selection condition of the form $\mathtt{O(A \sim c)}$ returns a set of objects of the form $o : s$ where s is the score of match for object o. We will denote this by $\mathtt{o : s} \in \mathtt{O(A \sim c)}$.

Given an atomic selection condition $\mathtt{O(A \sim c)}$, (1) if $\mathtt{A}$ is an attribute, $o : 1 \in O(A \sim c)$ iff $o \in DDB$ and o has type $\mathtt{O}$ and attribute $\mathtt{o.A}$ with value $\mathtt{c}$, (2) if $\mathtt{A}$ is a method then $o : s \in O(A \sim c)$ iff $o \in DDB$ and s is the score of the match assigned by method $\mathtt{o.A(c)}$.

A selection condition of the form $\mathtt{O(A \sim c)}$ where $\mathtt{c}$ is a selection condition then $o : s \in O(A \sim c)$ iff $s = f_{OR}(\{f_{AND}(s'', s') \mid o : s'' \in O(A \sim o'), o' : s' \in \mathtt{c}\})$.

The fuzzy combination functions `min, max` are logical choices here for the above functions which is what we will assume for this paper. However, any combination function can be used for satisfaction. Basically, suppose we are given a condition of the form $O(A \sim O'(B \sim c))$ such that $\{o_1 : 0.2, o_2 : 0.7\} \in (O'(B \sim c))$ and $\{o_3 : 0.8\} \in O(A \sim o_1)$ and $\{o_3 : 0.4, o_4 : 0.6\} \in O(A \sim o_2)$. Then, we can return o_3 either through its similarity to o_1 or o_2. As a result, the overall score for o_3 is given by $f_{OR}(f_{AND}(0.2, 0.8), f_{AND}(0.7, 0.4))$ which would produce 0.4 with respect to the `min, max` functions.

We define a *condition formula* recursively as follows:

- An atomic selection condition p is a condition formula.
- If cf_1, cf_2 are condition formulae, then so are $cf_1 \wedge cf_2$, $\neg(cf_1)$, and $cf_1 \vee cf_2$.

The satisfaction of condition formulae is defined in the usual way. Recall that selection conditions return objects with a match score. In the condition formulae, we assume the fuzzy and/or/not combination functions for the objects. For the purposes of this paper, we will use the *min, max* functions for the and/or combinations, however, other combination functions can be used based on the needs of the specific application.

B. Activities

Given the above model for objects, we would like to define an activity as a general concept that ties various different data objects. It is reasonable to consider an activity as a view defined over the data objects. However, it is important to model that not all objects are related to an activity in the same way given the specific meaning of an activity. For example, given a "paper writing" activity, the paper itself may be considered more relevant to an activity than the reservations we made to attend the conference the paper was presented in. As a result, we define the activity as an ordered (fuzzy) set where objects are considered more or less relevant to the activity based on a measure. An object may be relevant to many activities or none. The following definition simply states this fact.

Definition 3: (Activity) Suppose $\mathcal{O}$ refers to the universe of objects that could be stored in the computer. Then, an activity is defined as a function $\mathtt{actF}_{\preceq} : \mathcal{O} \rightarrow D_\tau$ where $\tau = (D_\tau, \preceq)$ is any partial order.

In the following we will omit the footer $\preceq$ in activity names, unless it is needed to distinguish among different cases. We will use the convention such that if $\mathtt{actF}(o_1) \preceq \mathtt{actF}(o_2)$ then o_1 is more relevant to the activity than o_2.

Now, given that activities are defined as a (partially) ordered set of objects with possible scores, we come to main problem being addressed in this paper. How do we define and use activities in an information system? We foresee the following main functionality at the core of management of activities:

- We develop a notion of an activity schema that allows users to define which objects are related to the activity based on their properties. This is very useful for bringing together objects that do not have any commonality other than the activity schema definition. However, the main problem is that most users are not familiar with query languages or even all the data stored on their computer to be able to create good models. Some users do not even create folders to organize their files.
- To address the possible problems associated with the first approach, we also assume the existence of methods that parse the data into different activities. We give an example of such a method in the following sections. The idea is that the computer can use the stored data to infer the most effective selection conditions in differentiating between the files on the computer.
- Given the two above methods, we need ways to help user manipulate her own data. For example, automatically generated models can be merged with user models to help users refine their schema. Different search queries can be mapped to activities to show the most relevant objects for that specific activity. Activity schema can be modified on the fly by changing the priority of different selection conditions to put emphasis on specific objects. We discuss these operations in detail in the next section.

C. Defining activities through query expressions

We first introduce ways to define activities using query expressions over the database of objects DDB_t at some time point t. Our intent is to define the set of relevant objects based on the properties they may have. If we are asked to specify what types of objects are relevant to an activity, we might want to specify a number of selection conditions over the properties of the objects. However, we might also be interested in specifying how related each condition is to the activity. The space of possible query languages for specifying this notion of relatedness is very large. We note two complementary approaches: one based on scores [13] and one based on preferences [5]. In the preference based approach, a constraint $cf_1 < cf_2$ is interpreted as that objects satisfying

cf_1 should be ranked strictly higher than those satisfying cf_2 (unless the object satisfies both conditions). In the score based approach, a degree of relatedness (also called degree of interest [13]) of each condition formula cf_i to the activity is defined independently of the other condition formulae. Both approaches have their problems. To fine tune the preference method to impose an ordering on most of the objects, we need to specify a very specific schema. This is very hard to do in our case. In the score based approach, it is hard to specify the correct scores to provide intuitive results. In this paper, we choose the score based approach where we compute scores implicitly as discussed in the following sections. However, the best schema model would be one that integrates both methods. The integration of partial order preferences to our schema is one of our future work directions.

Definition 4: (Activity Schema) Suppose $\mathcal{O}$ refers to the universe of objects that might be stored in the computer. Then an activity schema actS is given by a list actS $= \langle\, cf_1 : d_1 \ldots, cf_k : d_k \,\rangle$ where cf_i are arbitrary condition formulae and $d_1, \ldots, d_k$ are constants in $(0, 1]$ corresponding to the *degree of relatedness* of each condition formula.

Given $cf_i : d_i, cf_j : d_j$ in actS, if $d_i > d_j$ then any object o_1 that satisfies cf_i is more related to activity actS than any object o_2 that satisfies cf_j if $o_1 \neq o_2$.

Intuitively, the activity schema is given by a series of condition formulae. All objects that satisfy the given conditions are deemed relevant to the activity. However, the more relevant objects will have higher degree of relatedness than the others. This is similar to the notion of degree of interest given by Koutrika and Ioannidis [13]. We do not include in our model interest in the absence of values that is used in this model for simplicity. For example, suppose we want to describe objects relevant to my `trip to japan` by an activity schema as shown below:

DoR	Condition Formula
0.8	`ALL(about ∼ Interval('5/1/05-5/15/05')) ∨ File(directory ∼ 'japantrip') ∨ ALL(keyword ∼ 'japan trip kyoto')`
0.4	`Mail(keyword(sender) ∼ 'travelCo') ∧ Mail(keyword(content) ∼ 'itinerary')`
0.2	`Mail(sender ∼ Person(any ∼ 'keiko'))`

In the above schema, the keyword ALL refers to any object in the computer. The method about applies to any time attribute and assigns a score of 1 (highest) to the objects if they have any time specific property (creation and modification time) that falls in the given time interval. The score of objects at other time values are given as a function of the distance between their time property and the closest boundary of the given time interval. For example, an object created in January 1, 2005 will have a distance of 4 months to the given interval. The higher the distance, the lower the score of this object. The keyword(content) method is identical to the keyword method by applies only to the given property of the objects.

According to the schema above, the most relevant objects should have a time attribute in a specific interval, which is when we were on the trip. For example, pictures we took will have timestamps in this range. Also, other relevant information saved in a specific directory or with specific keywords are the most relevant. The second set of related objects are the mails corresponding to our plane tickets, so the mails corresponding to this trip are chosen. Finally, after the trip, we exchanged mail with a person we met during the trip. So, mails with this person are also considered related to the trip. Note that for the items with second and third priority, we did not explicitly define a time frame. However, we expect that the plane ticket purchase is at a time before the given interval and the mail exchange is after. While it is possible to specify these constraints in the activity schema, we can also use the natural ordering of time to help us derive these relationships as discussed in the next section.

Given an object may satisfy multiple condition formulae with different degree of relatedness, how do we find how related an object is to the overall activity? There are multiple ways to interpret this condition. One way is accumulative, the more conditions an object satisfies, the higher its relevance. Other interpretations are also possible as outlined in [13]. The following shows this interpretation of satisfaction.

Definition 5: (Activity defined by a schema) Suppose $\mathcal{O}$ refers to the universe of objects that might be stored in the computer. Given an activity schema actS $= \langle\, cf_1 : d_1 \ldots, cf_k : d_k \,\rangle$ and any object $o \in \mathcal{O}$, the tuple actS$(o) = \langle\, s_1 : d_1, \ldots, s_k : d_k \,\rangle$ represents the representation of object o in activity actS such that eihter $o : s_i \in cf_i$ or $s_i = 0$.

Then, the activity actF defined by schema actS is defined such that for any object $o \in \mathcal{O}$, actF$(o) = g($actS$(o))$.

One example function g is given by:

$$g(\langle\, s_1 : d_1, \ldots, s_k : d_k \,\rangle) = 1 - \Pi_{i=1}^{k}(1 - s_i * d_i)$$

For example, suppose in the schema given above, an object satisfies condition 1 and 2 (with degree of relevance 0.8 and 0.4) with scores 0.3 and 0.7 respectively. Then, the overall degree of relevance will be given by $1 - (1 - 0.3 * 0.8)(1 - 0.7 * 0.4) = 0.9328$. Instead, an object that satisfies only the first condition with relevance 0.8 will return the relevance $1 - (1 - 0.8 * 0.8) = 0.64$.

Other combination functions may also be adopted for this purpose based on how well they perform for a specific application.

D. Defining activities through objects

The first approach tries to define activities through the properties the objects might have. As we discussed earlier, the problem with this approach is that most users find it hard to define data models. Some users are just not interested. To this end, we decided to implement methods to extract activities automatically. There is some work in this area especially in extracting activities [15] or topics [1] from emails. The work of Khoussainov and Kushmerick is supervised. Given an example mail for an activity, they use Latent Semantic Indexing (LSI) to find all mails with similar keywords. The

work of Antonelli et. al. instead tries to find topic boundaries based on rules defined for a specific application domain. Our approach differs from those as it is an unsupervised method that uses a simple assumption. The actions of a person within a short time frame are highly likely to be related to the same activity. These actions can be editing files, writing emails, viewing files, visiting web sites, etc. Then, the content of the data files modified or accessed by these actions is also related to an activity. In essence, we treat the stream of data being produced by a person as a single long text and we try to find changes in topic, which is a well studied problem in information retrieval and natural language processing [16]. By finding topic boundaries, we can segment the text into smaller document vectors containing related items. Then, a person's many actions provide us with vectors with possible overlap. We treat this as a document collection and apply text retrieval methods to analyze it. Instead of LSI, we decided to use Latent Dirichlet Allocation [3] which models the documents as random mixtures over latent topics. Each topic is given by a multinomial distribution over the terms. A term may belong to multiple topics or none of them. We found that this model provides a very useful start for our purposes since we interpret each topic as a different activity and terms in each topic as the defining properties of that activity.

Given a series of actions $A_1, \ldots, A_n, \ldots$ a user is involved in, we compute a series of vectors $v_1, \ldots, v_n, \ldots$ that correspond to the properties of the object the action is accessing. For example, if the user is reading or editing a file, the keywords in that file are processed. Keywords are stemmed in the usual way. Furthermore, triples corresponding to the name, location, author of the file are also extracted. For an email, similar keywords for all the words in the text and the attachments are extracted. We also extract the people mentioned in the to, from, cc fields. Given that we are interested in activities, not all emails are related to an activity that we are involved in. For example, we might get mass emails containing announcements that are not related to our daily activities. This list of course includes countless junk email that might pass through spam filters. However, if we reply to an email that we receive, then it becomes related to an activity we are involved in. As a result, we only process mails that were sent as reply to a message from us. Given this model, we process the document vectors for each action in the order they occur within a time window of given size. We use a binary representation for the vectors where $v_{i,j} = 1$ if term j occurs in document i and 0 otherwise.

Given a time window of size k, we look at the document vector given by $W[i, i+k-1] = v_i + v_{i+1} + \ldots + v_{i+k-1}$ and check whether there exists any terms in $W[i, i+k-1]$ that occur at least m times where m is referred to as the minimum support. If so, then we store the vector $W[i, i+k-1]$ to our database as a new vector. Otherwise, we discard this vector. We shift our time window by one and continue processing in this way. The complete algorithm is given in Figure 1. We note that this algorithm produces possibly overlapping vectors that may contain unrelated terms mixed in with related terms

```
Algorithm findVectors
/* find vectors for sliding time windows
of length k and add to output if they have
a term with minimum support */
01.  while (true) {
02.  let v_i be the binary vector for the action at time i
03.  W = v_i + v_{i+1} + ... + v_{i+k-1}
04.  if W has a term with value at least m then
05.      add W to the database docDB
06.  i++
07.  }
```

Fig. 1. Algorithm findVectors

resulting from the inevitable context shifting. Our hope is that even though context shift does occur, it does not occur with the same keywords whenever we are involved in a specific activity. If it does then the chances are that it is related to the activity. We expect that such records will not play a significant role in the result as long as we have a substantial collection to work with.

We have run our algorithm on one of our author's (Shawn Pearce) desktop database. In our tests, we included the emails and text files from the last year and a half. There were a total of 14000 files, mostly mails. We extracted 60,000 keywords from these of which 7500 were different email addresses. There were about 14 keywords per mail and 150 keywords per text file on the average. Almost all text files we used were latex files created by the author that were stripped off their tags. Since we do not expect users to always have data available for their activities as a stream, we associated each text file with the a single timestamp, their last modification date. We processed at sent and received mails from this time but included only the sent mails that were in response to one of our mails as discussed above. We have then used the Latent Dirichlet Allocation algorithm (LDA) package provided by Blei [3].

To analyze the results, varied both the window size and the number of topics to extract which is an input to the LDA algorithm. Given the result, the owner of the files examined each topic and marked any recognizable activity for that topic. If a topic had all or all but one keywords associated with an activity, then he assigned a single label for this activity. In most cases, this happened when a personal contact was mixed in with the other information. In all other cases, all matching activities were added as a label to each topic. In some cases, certain topics were classified as unknown and no label was assigned to these topics. Given this, we compute the following measure:

$$Accuracy = \frac{\sum_i (num\,labels\,for\,topic\,i)}{num\,labels}$$

The table in figure 2 shows the results of our method. We report also on how many topics had no label, single label and also the maximum number of labels for any single topic. We compare our methods to an object based method which treats each mail and text file independently. This case is marked as window size 1. We see that there is a significant gain

in accuracy with the use of a window. We also note that a large percentage of the labels assigned in this case had the title "spam" or "blogspam" on it. Even though these were successfully classified, it is clear that the content was not very interesting to the author.

For window sizes 8 and 16, we see that the accuracy of our algorithm is best for 10 topics and also there are no unclassified topics for this case. We found out that if we extract 25 topics, even though there are 11+ topics with a single label, these topics tend to contain a lot of single emails or files and fewer general keywords. Another thing to note that for 25 topics, there were 2 or 3 topics that had no specific meaning to the user. This was not the case for five or ten topics. We also saw that increasing window size lead to a loss in accuracy as well as to the creation of topics with four different labels instead of 3 for the smaller window. This is to be expected since as the window size increases, it becomes more likely that unrelated files may be combined to a single vector. But given the small change in two cases, we conclude that the LDA method seems to be rather robust to the changes in window size. We also tried our algorithm by adding a single property that was the actual date in months for each data object. However, this resulted in a reduced accuracy. One possible reason for this is that the distribution of the date property is not compatible with the assumptions of the LDA method which assumes the topics are infinitely exchangeable within the document.

Window Size	Topic Number	Accuracy	No Label	Single Label	Max Num Labels
1	5	2.2	0	1	3
1	10	2	6	1	3
1	25	2.31	6	4	4
8	5	2.2	0	1	3
8	10	1.6	0	5	3
8	25	1.63	3	11	3
16	5	2.2	0	2	3
16	10	1.9	0	5	4
16	25	1.65	2	12	4

Fig. 2. The performance of our algorithm

Figure 3 shows two example topics found by our method, ordered with respect to their scores. For both topics, the first two terms had significantly higher score than the rest. The two topics correspond to undergraduate classes on Software Design and Documentation (SDD) Databases (DBS) respectively. The owner of the files was an instructor for the first class and an assistant for the second class. The other author of this paper, Sibel Adalı was an instructor for the DBS class. There is an additional topic for both of these classes returned by LDA. The names of people and most of the domains are changed to protect the privacy of the people in the list. The emails in the topics correspond to students in each class for the most part. For the SDD class, the email address listed for the author also varies in both topics since his affiliation was different when he was teaching the two courses. Person4 above was guest speaker in the SDD course, and Person5 was another person contacted to be a guest speaker for the same course. "Construction", "Inception" and "transition" were major projects in this class. The word "pattern" refers to the user patterns taught in this class. For the database class, most of the words are quite straightforward. The keyword "cgwork" refers to the directory that contains the files for this class.

Topic 1	**Topic 6**
person1@domain1.com	sibel@cs.rpi.edu
grade	hw
shawn.o.pearce@domain2.com	homework
person2@rpi.edu	spearc
presentation	cgwork
exam	database
inception	user
person3@cs.rpi.edu	data
question	us
lecture	csci
transition	query
class	server
Shawn_O_Pearce@domain3.com	person8@domain6.com
construction	db
schedule	send
person4@domain4.com	type
submission	differ
test	implementation
course	create
feedback	provide
person5@domain5.com	generate
deliver	requirement
pattern	time
person6@rpi.edu	rpi
person7@cs.rpi.edu	tex

Fig. 3. Examples topics found in our tests

The tests show that it is possible to extract meaningful activities with our method. We intend to conduct more tests to evaluate our algorithm with different data sets and add different data sets to our tests. We also intend to investigate is the effect of the size of document collection in the quality of results. Based on this, we can tailor our methods to drill down for specific topics for a specific activity.

IV. Exploring activities

Given the above definitions of an activity, we are now ready to describe how these can be used to implement a set of new database methodology for activities. The first method we described was by means of a database view that describe which objects are related to an activity. The second method finds distinct topics where each topic contains a set of keywords with a weight corresponding to the importance of the term in that topic. We map each topic to an activity and construct a schema for that definition as follows. We simply interpret each term as a simple keyword query which returns a score based on how many times the term appears in a searched object. The weights are interpreted as degree of relevance and we combine the terms using our combination function g as discussed in Section III-C.

Given these modalities for organizing data exist, the next question is what we would like to do with these data. In this section, we define a set of basic functionalities that we are implementing as part of an activity browser system. This system allows users to define and store activities in various ways as discussed below. It also enhances the desktop search functionality of systems such as the Mac OS.X spotlight and Google desktop search systems. These systems allow users to locate any type of file containing specific keywords. We call this a search functionality.

For the following discussion, assume the desktop contents at the current time is given by DDB. Suppose also that the desktop contains a set of activity schema $\{\mathsf{actS}_1, \ldots, \mathsf{actS}_n\}$ that are either defined by a user or derived automatically from the data as discussed in Section III-D.

A. Find related activities

One of the challenges in querying data using keywords or any other method is the correct output method. If the amount of data returned by the search is too large, then we need to either group the results into groups using either the semantic information associated with the data or using a clustering algorithm. Our approach builds on top of this where we show the semantic groups for each related activity.

Query 1. *Find all activities related to my search query.* Given an ordered set S of objects satisfying a keyword query, find the most related activities with respect to this set. For each activity, compute a match score. For each condition formula of that activity, compute a specialized score.

Given an activity schema actS, we can compute the match by simply adding the score of all objects with respect to this activity. However, note that a keyword query also returns objects ordered with respect to a score that measures how well the objects match the given query. Hence, the scoring method must take into account both the score of the objects in the search query and in the activity. Other ranking methods may be based on popularity or coverage of objects. For a given condition formula, we combine the degree of relevance of that formula with the match score of the objects in S that satisfy this formula. Note that this provides us with a way to provide context for a search. Similarly, it is possible to organize files with this function as in the following example.

Query 2. *Organize my emails based on the known activities.* For each activity, compute a match score as in the previous query and place each mail in one or more activity based on how well they match the given activity.

Also given a set of objects given by a search query as in the above section, the second functionality is to derive new activities with respect to this set. The question to answer here is what the most common properties of the returned objects are. To this end, we use a method such as the one in Section III-D to extract new activities and present these to the user together with the already defined activities. The idea is to provide the user with new conditions that they can add to their activity definitions. In most cases, users may not be very good at finding the most useful conditions and terms. We assume a browser should show both user defined and discovered activities in an intuitive way.

Query 3. *Show me all related activities for a specific time/person/place/file.* We can search and find all objects for a specific time, person or place, and then execute the same query as above. This allows us to find all related information for the queried concept classified under different activities.

Query 4. *Go back in time using the time-machine.* For the given search query and activities, add a new condition based on the date objects were created. By changing this condition, we are able to change the objects we are interested in. What would this query return two months ago? Given the activities that are in my system, which files that I had a year ago are relevant to these activities? This makes it possible to visualize the interesting relationships between our past and present activities.

B. Order objects with respect to an activity

Given a set of objects, order them with respect to their score in an activity. This allows us to order objects that satisfy a specific condition with respect to their relevance to an activity. This functionality requires simply to compute the scores with respect to the activity and then sort the objects with respect to the given score. We note two variations of this functionality.

Query 1. *Show me the files on the visit to Company Acme last year.* Find the dates, people involved in the visit, files created for the trip and organize them in the order of relevance together with the relevant categories of information.

This assumes the visit is an activity actF_i defined by actS_i. To compute this query, we simply return all the files in DDB in the order of actF_i.

Query 2. *Limit my keyword search to those items relevant to activity "Writing the activity paper".* Order the matching items with respect to their match to the given activity.

Find all objects that satisfy a given search query and their relevance to the given activity actF_i. Given a preset threshold ϵ, remove from the interface all objects with relevance less than ϵ to actF_i.

Query 3. *Hide all items relevant to activity "Car Purchase" in all my searches.* Given a level of sensitivity, do not show the items that appear to be related to a specific activity. For example, in a professional setting, do not show files related to personal use of the same computer.

Given the threshold θ and an activity actF_i marked as hidden, remove all objects from the interface with relevance of θ or higher to actF_i. This query shows how we can implement privacy on top of our system. It is easy to classify activities into "work" and "fun", and simply hide fun activities while at work.

C. Alter activity schema

Very often, a search query specifies what we are searching for. However, we might also be interested in specifying the actual context of our query. For example, suppose we are looking at files ordered with respect to a specific activity, say buying a house. However, we are interested in mostly the

house search aspect where we want to locate a house that we had seen but we do not remember which one. We can narrow down the house by emphasizing in the schema the conditions related to this house, suppose a location and a type of house. Assuming these conditions already exist in the schema, we can simply change their degree of relevance by manipulating their ordering with the help of a user interface. This changes the overall relevance of documents, emphasizing the information we had in mind.

Query 1. *Show me all the house information related to the house buying activity but put emphasis around dates May 2005.* Assuming that we remember that we saw some houses around May 2005 that we liked for different reasons. Hence, we want to put an emphasis on houses that satisfy this condition. Note that this is a bit different than the previous queries on ordering objects. We do not want to keep the ordering of objects with respect to the selection condition `ALL(date` $\sim$ `05/2005)` since we would like to use the ordering given by this activity. However, we do not want to simply return the ordering of this activity either. Instead, we would like to combine this condition into the activity schema and then change the priority of different conditions to highlight the objects we are looking for.

D. Merge activity schema

One of the important functionality that our browser integrates is the modification of schema. It is possible for users to modify schemas on the fly by merging conditions returned by the system into existing schema or create a new activity from an automatically generated category. However, for this functionality to be useful, it should allow for the merging of two different schema regardless of their origin. Implementing such a method requires that we address two main challenges. Suppose, we are merging $\mathsf{actS}, \mathsf{actS}'$. First, we need to find equivalent condition formulae in both schemas so that we do not repeat information or give a higher degree of relevance to repeated information. Recall that our combination function favors objects that satisfy more condition formulae in the schema. However, finding semantic equality or subsumption relationships is a very hard problem to solve in general. Instead, we use simple but incomplete checks for equality. The second problem is the correct way of integrating the degree of relevance from either schema. In the absence of any specific information, we would like to treat both schemas as equal. It is possible also to integrate them by giving precedence to one of the schema.

To implement equal merge, we take two schema $\mathsf{actS} = \langle\, cf_1 : d_1 \ldots, cf_k : d_k \,\rangle$ and $\mathsf{actS}' = \langle\, cf'_1 : d'_1 \ldots, cf'_k : d'_k \,\rangle$ such that the minimum and maximum degree of relevance values in both schema are comparable. If they are not, we assume the degree of relevance values in a schema can be scaled by dividing all the values with a constant. The merged schema $\mathsf{actS} \oplus \mathsf{actS}'$ then contains the union of the both conditions with the following exceptions. If condition cf_i is equivalent to cf'_j, then we only store one of them with the degree of relevance $(d_i + d'_j)/2$. If condition cf_i subsumes condition cf'_j then, we disregard cf'_j. In this case, we modify the degree of relevance only if $d'_j >> d_i$.

E. Schema querying

In addition to the merge functionality defined above, it might be useful to add other methods to query schema. Examples of such methods are described in this section. One possible functionality would be to cluster schemas to find similar schemas and merge them. Without very detailed information about the schemas, it is hard to implement this functionality solely based on the schema definitions. Instead, it is possible to consider how they relate to the objects stored in the system. One approach would be for each given activity, find the ordering of objects and then order all the other activities with respect to this ordering. This allows us to construct a graph where each node is an activity and each directed edge is a measure of how close the activities are. This graph can then be used for clustering activities into general groups.

Figure 4 shows a snapshot of the browser system for a search query. Whenever the user initiates a search query, it is possible to invoke this interface to see the most relevant activities, for each activity, the most relevant conditions and the most relevant objects. It is possible to retrieve all objects for a category by clicking on that category. It is possible to view all objects for a category by clicking on the "see more" link. The sliders on the right show the most relevant properties of the objects being shown on the left hand side. By sliding these, the user is able to change the degree of relevance values for the given conditions for this query.

V. Conclusions and Future Work

In this paper, we defined the notion of an activity as a way to group objects in a user's desktop into overlapping clusters of related objects and related properties. Our work is a first step towards solving the problem of scale when dealing with an ever increasing amount of data both on our own desktop as well as in other data sources that we use and share. Even though available semantic information such as free text or semantic annotations can be consumed easily in any desktop system including ours, generating this information is still very resource intensive. Similarly, content-based retrieval methods for image, video and other media suffer from the problem of being too general. The content of an image may be described very differently based on context. Hence, there is a need to integrate these methods with other data organization methods such as activities to facilitate their effective use.

We are in the process of implementing our prototype activity search and browse system as described in this paper. To this end, we are investigating various algorithmic and system issues in the implementation of this system. One of the main future problems we need to address is the issue of structured activities where an activity may be described by combining simpler activities. Even activities may have many different aspects, for example a trip has a preparation phase, the actual trip followed by the other related activities. Based

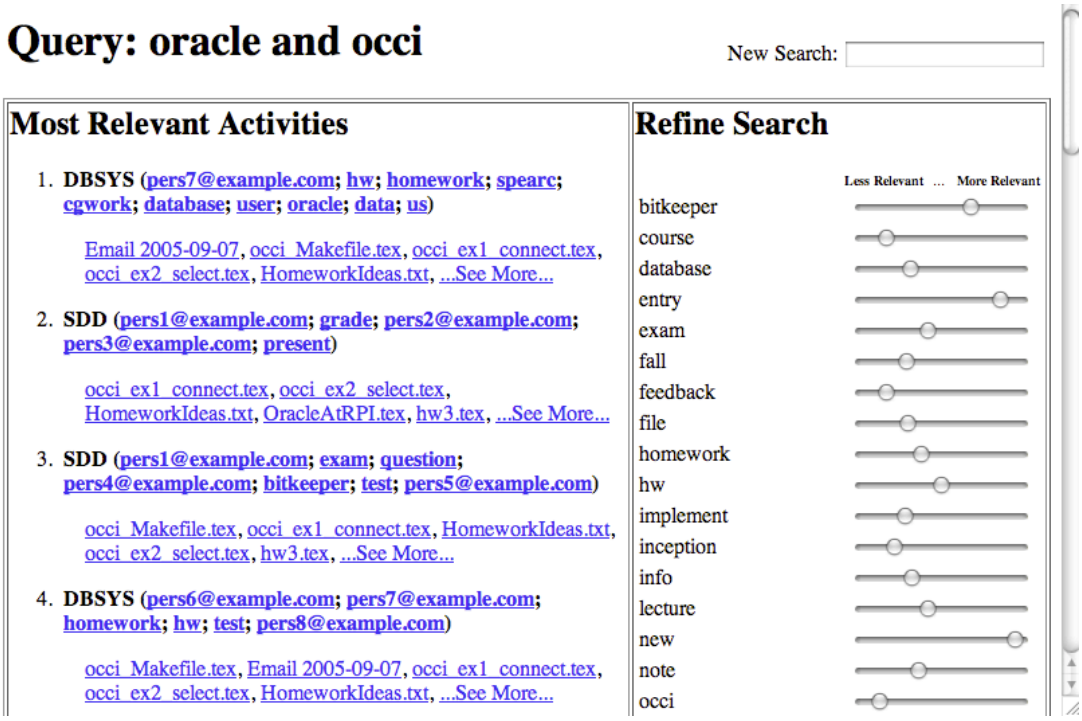

Fig. 4. An example screen capture of our browser application

on our queries, we might be interested in a certain aspect of a given activity and the system should immediately adapt to this using a form of relevance feedback. Even though we can keep activity definitions fairly simple, we can learn about user's specific preferences based on their interactions with the system and integrate these back into the system. Our long term goal is to augment the desktop with inference tools that make use of the semantic data available in the activities to automatically associate semantics with data objects. For example, we can generate new rules to find time and location information associated with objects based on the way the similar time or location information appears in different formats, following a set of related properties and etc. The availability of these solutions would be a an important first step towards solving the problem of scale in information systems.

Acknowledgment. We would like to thank Ramesh Jain for stimulating discussions on multimedia querying and experiential computing. We also would like to thank Bouchra Bouqata for helpful discussions.

REFERENCES

[1] F. Antonelli, J. W. Kim, K.S. Candan and M.L. Sapino, "Navigation Support for Students who are Blind in Accessing Discussion Boards" , in *Proceedings of CIAH05*.

[2] P. Appan, H. Sundaram, and D. Birchfield, "Communicating everyday experiences", in *Proceedings of the 1st ACM workshop on Story representation, mechanism and context*, 2004.

[3] D. Blei, A. Ng, and M. Jordan. "Latent Dirichlet Allocation", Journal of Machine Learning Research 3(2003), pp. 993–1022.

[4] S. Boll and U. Westermann, "Mediaether: an event space for context-aware multimedia experiences", in *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence*, 2003.

[5] J. Chomicki, "Preference formulas in relational queries". ACM Trans. Database Syst. 28(4): 427-466 (2003).

[6] Vision of Chandler: *www.osafoundation.org*, Consulted in May 2005.

[7] A. Cheyer, J. Park and Richard Giuli. "IRIS: Integrate. Relate.Infer. Share." in *Proc. of the First Workshop on Semantic Desktop - Next Generation Information Management & Collaboration Infrastructure*, pp. 64–78.

[8] X. Dong and A. Halevy, "A Platform for Personal Information Management and Integration", , in *Proceedings of CIDR*, 2005.

[9] S. T. Dumais, E. Cutrell, J. J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. "Stuff i've seen: A system for personal information retrieval and re-use". in *Proceedings of SIGIR*, 2003.

[10] E. Freeman and D. Gelernter. "Lifestreams: a storage model for personal data". in *SIGMOD Bulletin*, 1996.

[11] R. Jain. "Experiential computing", Commun. ACM, vol.46(7), 2003, pp. 48–55.

[12] D.R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. "Haystack: A General Purpose Information Management Tool for End Users of Semistructured Data". in *Proceedings of CIDR*, 2005.

[13] G. Koutrika and Y. Ioannidis, "Personalized Queries under a Generalized Preference Model", in *Proceedings of ICDE 2005*.

[14] D. V. Kalashnikov, S. Mehrotra, and Z. Chen. "Exploiting Relationships for Domain-Independent Data Cleaning". in *Proceedings of SDM*, 2005.

[15] R. Khoussainov and N. Kushmerick, "Email Task Management: An Iterative Relational Learning Approach", in *Proc. Int. Conf. Intelligent User Interfaces*, 2006.

[16] C. D. Manning, and H. Schutze, "Foundations of Statistical Natural Language Processing", The MIT Press, 1999.

[17] "MyLifeBits Project", *research.microsoft.com/barc/mediapresence/MyLifeBits.aspx*, 2005

[18] "Resource Description Framework" (RDF) *www.w3.org/RDF/*, Consulted in May 2005.

[19] R. Singh, Z. Li, P. Kim, D. Pack, and R. Jain, "Event-Based Modeling and Processing of Digital Media", in *Proc. of First International Workshop on Computer Vision meets Databases, CVDB2004*.

[20] L. Sauermann. "The Semantic Desktop, a basis for Personal Knowledge Management". in *Proceedings of I-KNOW 05*.