Title: Incremental Visual Queries

Author: Mark Derthick

Address:

Human Computer Interaction Institute

Carnegie Mellon University

Pittsburgh, PA 15213

(412) 268-8812 (telephone)

(412) 268-1266 (fax)

mad@cs.cmu.edu

Submitted to ACM Transactions on Information Systems (TOIS)

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Graphical user interfaces (GUI), Interaction styles.

H.2.3 [**Languages**]: Query Languages.

Abstract

Exploratory data analysis is a human-centered process whose goal is to develop intuition and insight into a dataset. Usually initial findings will suggest new questions, leading to extensive iteration. Statistical graphics play an important role, but a flexible means for populating them with relevant data is also needed. Database query languages are one means, but formulating queries is a distraction from the primary task of understanding the data. This paper uses the concept of *isets*, which embody both a description and a set of entities that satisfy it, to bridge the gap between the expressive power of queries and the concreteness of data. During exploration, isets can be incrementally modified, related to other isets, or used to define new isets. This last operation uses the metaphor of navigation from an existing iset as the means of arriving at the new related iset, which situates the analyst as actively moving through the world of the data. The navigation path among isets resembles a graphical query language. However, visualizations give continuous feedback as the graph is incrementally constructed, rather than only after an "execute" command. Navigation and nine other operations are meant to effectively support iterative exploration.

1. Introduction

1.1. Motivation

Our goal is to enable analysts to explore large relational datasets, communicate their findings, and make decisions using direct manipulation [Shneiderman 1982] visualization-based interfaces, whose high-bandwidth perceptual/motor channels minimize cognitive overhead. Research in exploring databases has been dominated by automated machine learning methods, although it is widely acknowledged [Brachman et al. 1993; Fayyad et al. 1996] that much of the work involves humans, who must assemble and clean the dataset, guide the learning algorithm, and interpret its findings. Usually these findings will suggest new questions, leading to an extended iterative process whose deliverables are presentations of the patterns found. These in turn support decision-making.

Incremental exploration should build on previous intermediate results, modifying them in ways that analysts find logical. This paper defines ten query transformation operations that correspond to logical modifications: navigation, partition, filtration, and projection; set union, difference, and intersection; copy and clone; and brushing. Transformations are applied by direct manipulation of queries, which are expressed in a visual language that integrates summary visualizations of the results. In some cases, multiple transformations are applied automatically in anticipation of user needs, giving rise to multiple result sets at once. Intermediate result sets are visualized in universe/result pairs, which support part-whole comparison and provide a visual history of the exploration process.

SQL also has about ten syntactic query constructs. If the proposed set of query transformation operations requires fewer steps to iteratively explore data than syntactic SQL modifications would, they constitute a better theory of interactive exploration. For instance, in SQL, the equivalent of navigating from a set of contacts to the set of email messages they received would involve either 1) adding the email table to the `FROM` clause as well as join constraints to the `WHERE` clause, projecting back down to the attributes of email, and using `DISTINCT` or 2) using a nested `EXISTS` query. In contrast to the notion of relational completeness, which is based on the *expressiveness* of a language for posing useful queries, we might consider "interactive completeness" to require a language that is *effective* for useful query *sequences*. Distinguishing expressiveness (can it be done) from effectiveness (can it be done easily) in the HCI context is due to Mackinlay [1986] .

Most previous data mining and visualization systems and theory apply to a single table or view of data [Velleman 1993; Card et al. 1999; Wilkinson 1999; SAS Institute 2001; Silicon Graphics Inc. 2004], and can select subsets of rows (filtration) and columns (projection). However databases commonly have multiple related tables, and queries commonly

employ `JOIN` (roughly like navigation), `GROUP BY` (partition), [set] `UNION`, and `MINUS` (set difference). Even non-experts regularly use phone, PDA, or desktop software to manage related tasks, contacts, email, and phone calls. Email is regularly used for task management [Whittaker and Sidner 1996], but messages are often at the wrong granularity. One message may relate to multiple tasks, and a task may relate to multiple messages. The same many-to-many relationship holds with contacts. Current interfaces poorly support summaries over relational data like "how long did I spend on Project X, and who were my main contacts?" Beyond the useful task-specific query types that are built in to an application, there is a need for task-tailorable extensions [Neuwirth et al. 1998]. Both for expert analysts going through the full cycle of cleaning to reporting, as well as for more casual users iterating over a few queries, summarizing multiple sets of relational data adds useful expressiveness compared to single-view systems.

1.2. Terminology

Regrettably, the fields of databases, information retrieval, and user interfaces do not use terms consistently. This paper primarily adopts database jargon, and in particular the **entity relationship** conceptual data model. For instance, email is a type of entity, and sender is a relationship between instances of email and instances of contacts. An instance of email also has **attributes** such as `date_sent` and `body`. Attributes can be either primitive (in the sense of KLONE [Brachman and Schmolze 1986]) or derived. A derived attribute has either a declarative definition or an attached procedure that computes its value. An instance of a relationship is a tuple of entities, which in our system is itself a first-class entity. This allows every visual object to represent an entity. There has been no need for tuple entities whose elements are tuple entities.

Isets (short for "intensional sets") don't quite correspond to any existing term. They are first class entities with both an intension and an extension. The intension is a description that applies to each element of the iset, and functions like a query (e.g. "contacts who sent me email yesterday"). The extension is like a result set or view (e.g. {Bob, Sue}). The extension can change over time, either because updates change which elements satisfy the description, or because the description itself changes. In a direct manipulation interface, where input = output [Shneiderman 1982], it is natural to merge the notions of query and result. Furnas and Rauch's e-sets [1998] are similar, except that they are tied to an interface widget. "Aggregate" has been used in the information visualization literature [Goldstein et al. 1994; Rayson 1999; Stolte and Hanrahan 2000], but can be confused with the database meaning of a physical level table used to store incremental results. "Collection" is sometimes used in the database literature, but in the information retrieval literature it connotes long-lived document sets.

There is a sense in which the iset intension is primary, since it determines the extension. At the interface level, though, it is compelling to drag objects into or out of a visualization of iset members in order to change the membership. Supporting user interaction at either level, isets bridge the gap between the expressive power of a query and the concreteness of data. If the extension changes (e.g. {Bob, Sue, Fred}), adding a `Where` expression like "`Or ID in (Fred_001)`" to the intension restores consistency. If the intension is changed (e.g. "contacts at CMU who sent me mail yesterday"), the extension is updated by removing the non-CMU contacts.

1.3. Roadmap

In previous work, we presented a small yet powerful set of Direct Manipulation interface operations for exploring small datasets, where objects are visualized individually, and an implementation in Visage [Roth et al. 1997]. Later, we described the Visual Query Environment (VQE), which extends Visage operations to aggregates of many objects [Derthick et al. 1997]. This paper concentrates on the principles behind the VQE interface and its operations. Its data model attempts to match the user's domain model so that the tool is a natural extension of the user's thought process rather than a barrier between data and understanding. The operations are all transformations on datasets. In the emerging terminology of the data visualization pipeline from raw data to rendered graphic, they have been called "data transformations" [Card et al. 1999; Tang et al. 2003], "data functions" [Wilkinson 1999], and "within analytical abstraction operations" [Chi 2000].

1. The next section explains **the ten operations,** the idea of **iset pairs**, and how **derived attributes** magnify the expressiveness of the operations.
2. Following that are more detailed examples that also cover implementation issues.
3. Section 4 gives a **formal definition** of the operations by mapping them to SQL.
4. Next is a description of **related work** in visualization theory and practice.
5. Last are a discussion of **future work**, **evaluation**, and a **summary**.

Analysts require more than just exploration operations. Elsewhere we have discussed 1) integration of exploration and presentation tools [Roth et al. 1997], 2) the relationship between queries and record-level visualizations and the extension of brushing to visual objects that represent tuple entities [Derthick et al. 1997], and 3) supporting work over time by visualizing operations on base entities and creating reusable visual queries from subsets of these operations [Derthick and Roth 2001a; Derthick and Roth 2001b].

2. The Proposed Data Exploration Operations

The examples below use data from a personal digital assistant database, where the analyst is expected to be the PDA owner (in this case, the author — md). The database includes email (7000 messages), calendar (1000 appointments), contacts (500), telephone conversations (400), tasks (400), and document visits (100). This data is collected by Microsoft Outlook, and periodically downloaded into a relational database of about 4MB. Figure 1 shows the part of the schema used in the examples. Outlook can only make outgoing phone calls, so the caller (md) is implicit. Contacts can be related to email messages in three ways: as sender, recipient, or cc.

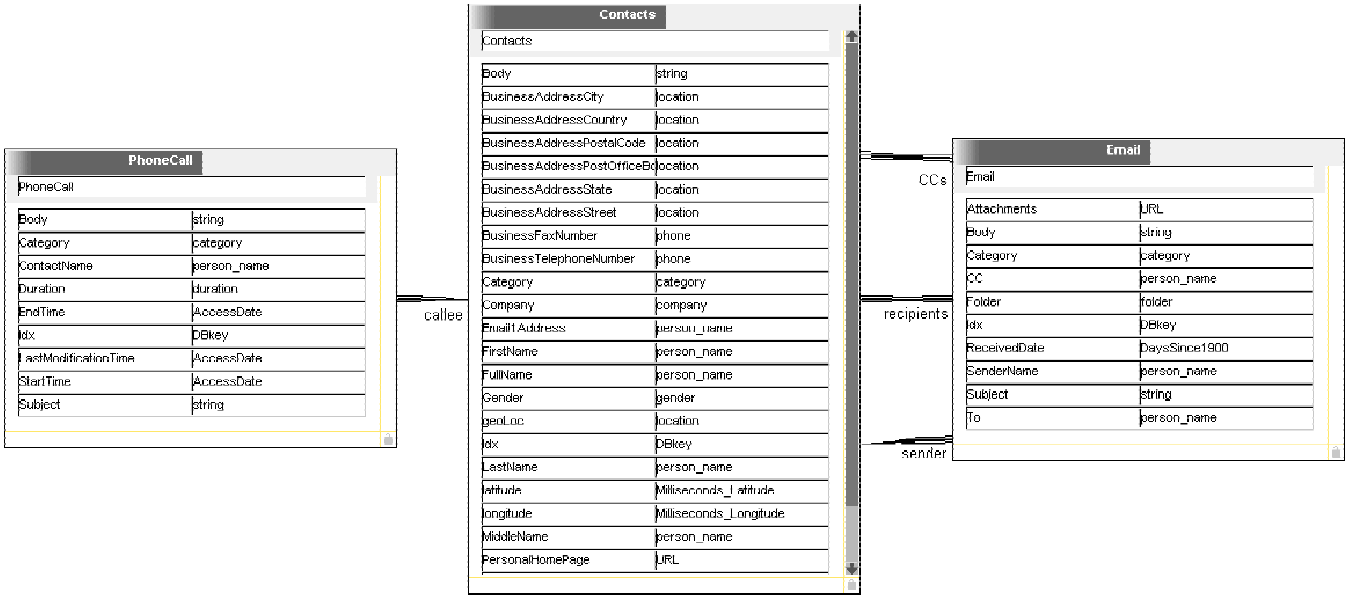

Figure 1 The subschema used for the examples below. For each of the three entity types, its attributes and attribute data types are shown. The links show the binary relationships that can hold between instances of the entity types. The link patterns indicate that recipients and CCs are many-to-many, while callee and sender are many-to-one.

Much recent research has focused on using this kind of data, sometimes augmented with continuously recorded audio, video, and GPS coordinates, to provide context for numerous tasks [Freeman and Gelernter 1996; Rhodes and Starner 1996; Abowd et al. 1997; Houghton 1999; Wactlar et al. 1999; Bell and Gray 2000; Hauptmann and Lin 2001]. Currently, visualization and summarization play little part in these tasks. For example, no current email system can plot the number of messages from employees of a given company over time. Our eventual goal in this domain is to discover the kind of queries that are most useful, and then to design simple information appliances to enable non-experts to use them.

2.1. Iset Pair Widgets and Filtering

Isets are abstract data entities, but it is easier to understand the effects of operations on them through a concrete visualization. Figure 2 shows the VQE widget used to represent an iset pair as it is created (left), as Dynamic Query (DQ)

[Ahlberg et al. 1992] widgets are dropped on two of the attributes (middle), and as they are used to filter the result iset (right). An iset widget supports querying on its iset. The iset serves as the universe from which results are drawn. Therefore, when a widget is created for an iset, it ensures that a paired iset is created to represent the results of the query.

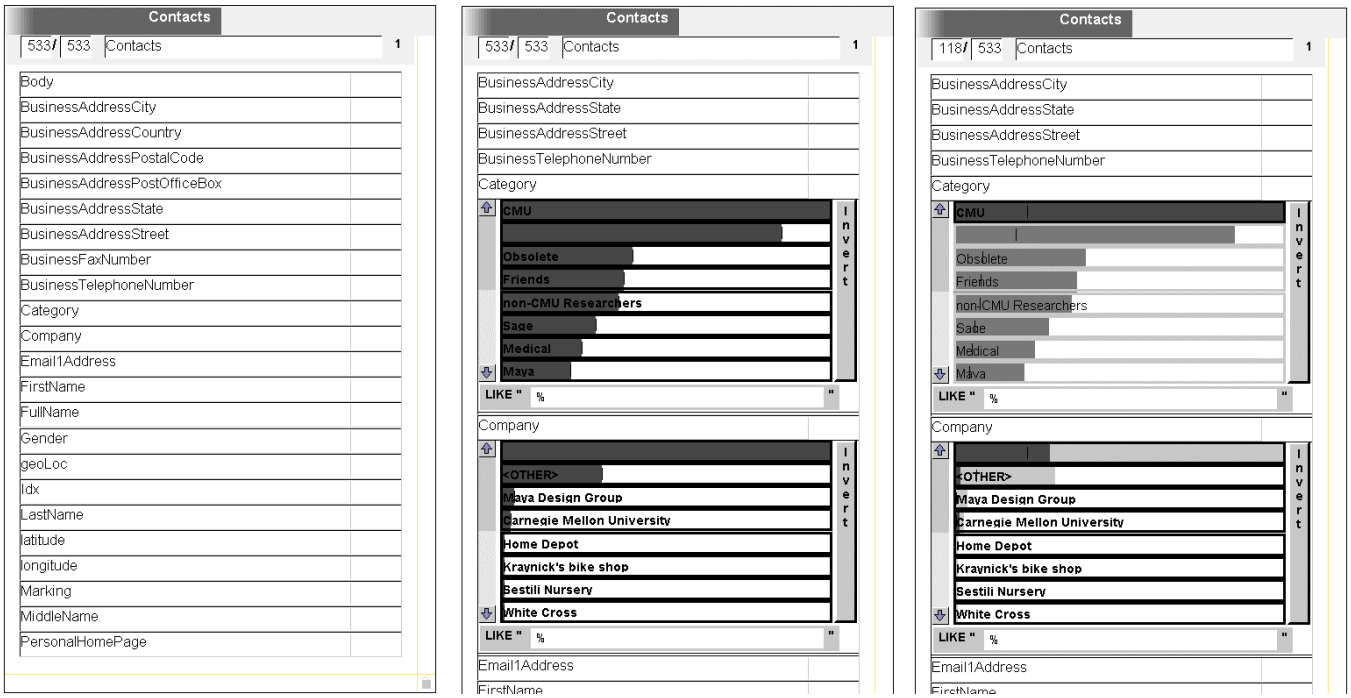

Figure 2 Three successive snapshots of an iset pair widget as operations are performed on the isets.

In an iset widget, the title bar can be edited by users to describe the universe iset. The default description is its entity type. The iset in Figure 2 includes all the contacts in the database, so this default description is appropriate. The summary line just below the title shows the cardinalities of the result and universe isets, and their entity type. Below that is a list of the attributes that entities of this type can have. In the middle view, the two DQ widgets show the distribution of the 533 contacts by category and by company. For instance, the most common value for `category` is "CMU" while the most common value for `company` is the empty string. When the members of the universe iset have more than 16 different values for an attribute, only the most common 15 are shown as individual histogram bars, and the rest are aggregated and labeled "<OTHER>." In this case, the full set of values can be searched using the SQL `LIKE` operator using the type-in box below the bars. <OTHER> is the second longest bar for company; it is the 15th longest bar for `category`, so a user would have to scroll (or drag the bottom edge of the DQ widget to make it show more bars) to see it in this figure.

On the right, all values of the `category` attribute except CMU have been deselected. (First the invert button deselected all values, and then clicking the CMU bar reselected that value.) The selection state of each value is shown by the border of each bar. Selected values have a black border, while deselected values have a light gray border. The result set

now includes only those contacts whose `category` attribute has the value CMU. The iset widget summary line shows that 118 of the 533 contacts satisfy this condition. The light bars in the histograms continue to show the distributions for the universe iset, while the dark bars show the distributions of the result iset. There is a vertical hash mark superimposed on each light bar |result iset| / |universe iset| of the way across (118/533 of the way across in this case). For each attribute, the hash marks show what the distribution of dark bars would be if the attribute were independent of the conditions imposed on the result iset. In this example, the condition involves only the `category` attribute, so it is no surprise that the actual distribution is higher than the hash marks for `cateogry` = CMU and lower for the other values. It is more interesting that for `company` there are more contacts with the value "" than there would be if `category` and `company` were independent. Apparently md does not bother to fill in a company when the category CMU already implies one.

Figure 3 shows in more detail the meaning of each part of an iset widget. This iset was created by copy-dragging the result iset from Figure 2 (right) to a VQE tool. Thus the same iset is shown as a result in one view and a universe in another. If further filtering were performed in Figure 2, say by reselecting the value `category` = Friends, the cardinality value 118 in Figure 3 would increase as well. (It is also possible to create a new iset by dragging instead of creating a new visual object referring to the same iset, as described in Section 2.6.) The user has changed the title of the widget to "CMU Contacts," dropped a DQ widget on the attribute `company`, and toggled off the value "".

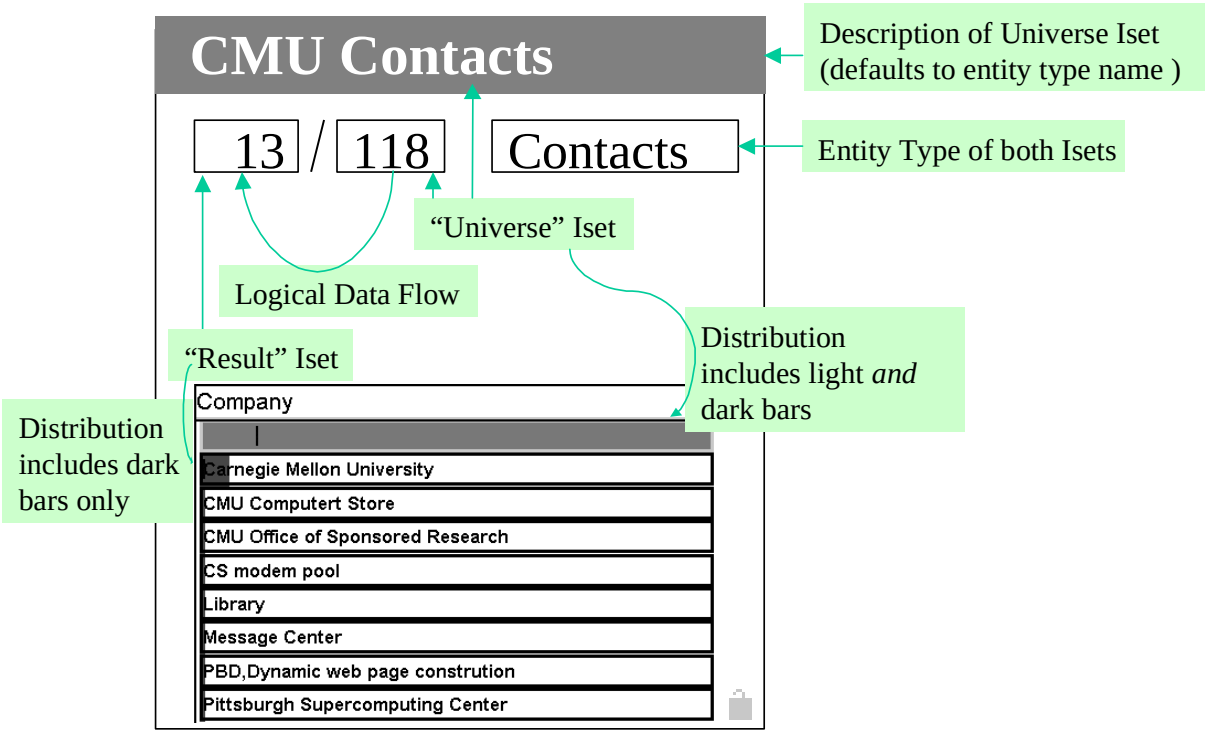

Figure 3 The visual representation of an iset pair. The universe iset contains 118 contacts, while the result iset contains 13.

This design satisfies three criteria that a good exploration interface should support: 1) refining or expanding an iset, 2) making the relationship between isets clear, and 3) returning to intermediate results:

- **Refining isets** Refining by strengthening (adding more constraints to) an iset's intension is such a useful operation that VQE automatically creates a paired result iset for each universe iset it displays. Filtration is such a useful kind of strengthening that affordances for dropping DQ widgets on each attribute are included as well.
- **Relating isets** The fraction notation clarifies the subset relationship between universe and result. Both the fraction and the juxtaposition of the light and dark histogram bars allow easy part-to-whole comparison.
- **Intermediate Results** are immediately accessible even in a long chain of query transformations because universe isets' definitions do not change during additional navigation, partition, filtration, or projection operations.

2.2. Navigate

Database interfaces such as Microsoft Access sometimes visualize joins between tables as a *query graph*, as in Figure 4 (bottom). In an entity-relationship schema, the nodes are entity types (`contacts` and `email`) and the link(s) between them are relationships (`sender_inverse`). In an interactive system, query graphs start with a single node, and additional nodes and links are added over time. As nodes are added, there is a simultaneous mental shift of focus, for example from `CMU Contacts` to the email they sent. *Navigation is our name for shifting mental focus from one iset or other entity to another related one.*

In Visage, navigation operations start at one individual entity (e.g. `Mary`) and proceed to another (e.g. `Message_001`) via a relationship (e.g. `sender_inverse`). In VQE, navigation starts at one universe iset (e.g. `CMU Contacts in` Figure 4 `(top)`) and *creates* another (e.g. `Email from CMU Contacts in` Figure 4 `(bottom)`) via a relationship (e.g. `sender_inverse`). The direction of the arrow always reflects the temporal analysis order: thus the arrow points from contacts to messages and is labeled "`sender_inverse`." This process is thought of as navigating in parallel from each `contact` to the `email` he/she sent. The union of all these sets forms the universe iset of the new iset widget. After the new iset is defined, the old and new are linked with a standard database inner join. Below we often use the descriptive terms "multi-entity query", "structured query", or "navigation iset" to represent the iset of entity tuples that satisfy the query graph as a whole, instead of the database terms "join records" or "join view."

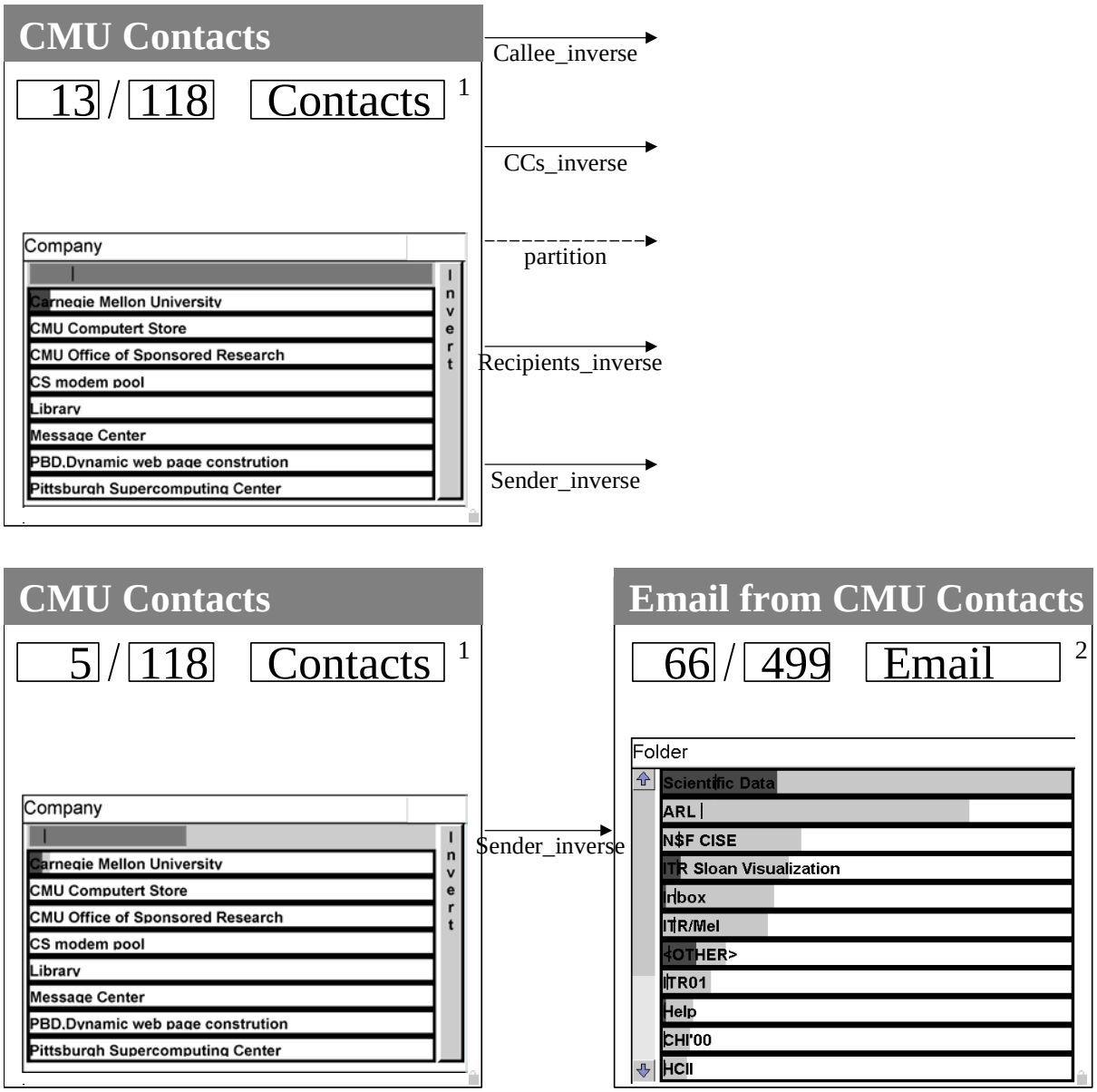

Figure 4 Top: Double-clicking on the background of the iset widget shown in Figure 3 exposes the navigation menu. Bottom: Dragging the sender_inverse arrow creates a new iset and locates an iset widget visualizing it. The two nodes and the link constitute a graph representing a multi-entity query.

The fact that the Contacts result set cardinality decreases from 13 to 5 indicates that the join is lossy; only 5 of the 13 contacts sent any email. The dark distribution over email folders reflects the 66 messages sent by the 5 result contacts with a known company. As shown by the hash marks in the email `folder` histogram, messages from these 5 were more likely to be filed in the Scientific Data, ITR Sloan Visualization, and <OTHER> folders compared to messages from CMU contacts in general, and less likely to be filed in other explicitly visualized folders. If the restriction on company were removed, the resulting contacts would account for all 499 messages in the email universe and the dark folder distribution would be the same as the light folder distribution. However, due to the lossy join, not all 118 contacts would be in the

contacts result iset. An estimate of the join selectivity is shown by the company histogram. In addition to very dark and very light bars, there is an intermediate gray that shows how long the dark bar would be if the value were selected. Since the medium gray `company = “”` bar is about 40% as long as the light gray bar, reselecting it would add about 40% of the contacts with no known company to the result iset. Tweedie et. al. [1996] were the first to use multiple histogram colors to indicate how many constraints must be satisfied before entities would belong to a set

This feedback contributes to the incremental nature of data exploration in VQE. Visualizations are continually updated as queries evolve over an exploration session. There is no distinct “execute” step that transforms a query into a result only after it is in “final” form. As exploration proceeds, each new iset is defined in terms of previous ones. Without the context sensitivity, it would be disconcerting to be looking at the `Contacts` for a `company`, navigate to related `email`, and find the dark `Folder` distribution almost invisible in comparison to the light distribution of all 6839 messages. To maintain this incremental nature, VQE actively augments isets as new entities become relevant: When new objects are dropped on an iset widget (and therefore added to its universe iset), VQE automatically navigates to related isets and populates them with related entities. For instance, if a new `contact` were created and dropped on the left iset widget in Figure 4, all the messages sent by that contact would be added to the right iset. The ability to propagate “forward” allows revising the original query and following the consequences. “Backward” propagation allows changing the destination iset (e.g. `email`) to see how the original query (e.g. `contacts`) would have had to be different to get those results. In order to override this propagation behavior, the lock icon that appears in the lower right corner of every Visage frame can be toggled on. No entities can be added to locked iset widgets.

Although less common than navigation, ordinary joins are also supported; the relationship arrow is dropped on an existing iset widget rather than the VQE background.

2.3. Project

VQE projection is just an object-oriented version of the relational database operation. From a navigation iset, projecting an entity tuple down to a single node in the query graph extracts the appropriate tuple element. In Figure 4, the relationship between the two universe isets and their result isets requires projection as well as filtration. The navigation iset (dotted rectangle at upper left of Figure 5) was created automatically as part of the navigation operation, but is not shown to users. The members of this iset are (`<contacts>`, `<email>`) tuples. Filtration operates on the navigation iset and the effect is realized in a paired result iset (labeled “‘Join’ Result” in Figure 5). If there are contacts who sent no email, there will be no tuples containing them. Contacts who sent many messages will appear in many tuples. Therefore, the number of contacts

can't be counted just by counting tuples. Instead, the result of filtering is projected back down to each query graph node, and duplicate entities are ignored. Thus the semantic relationship between universe and result isets in a structured query involves navigation, filtration, and projection operations. Behind the scenes, there are now 6 isets being maintained. Four are the result and universe isets for the two iset widgets, but a result and universe iset are also needed for the `join`.

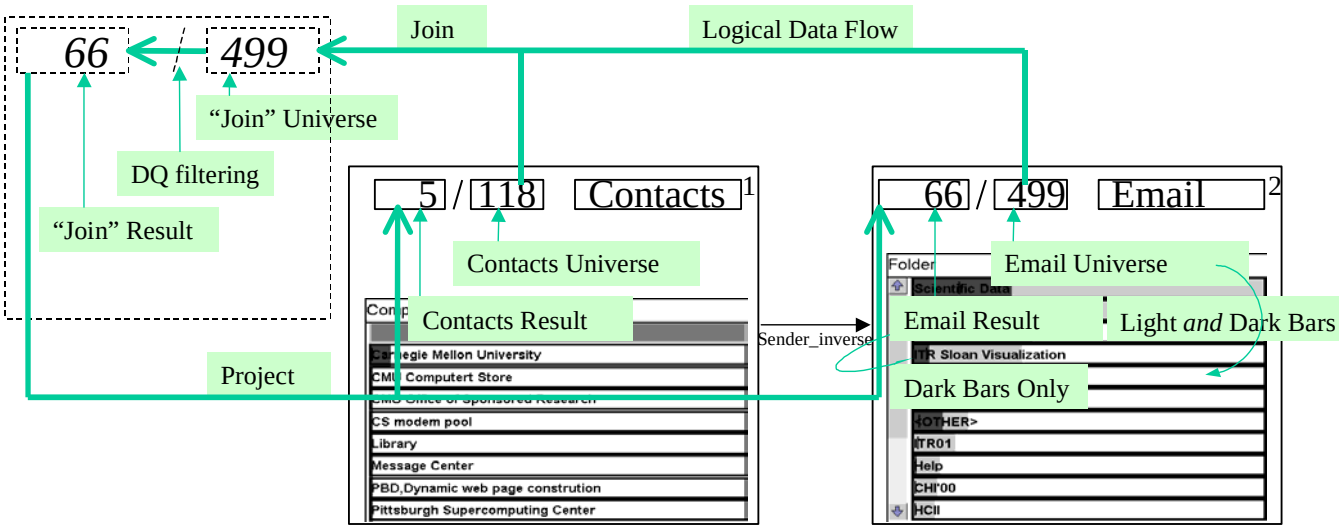

Figure 5 Data flow for a navigation operation in terms of the database operations join, select (for the DQ filtering), and project.

The DQ filtering operates on derived attributes of the tuple entities. The relationships `node_1`, `node_2`, etc. link each tuple entity to its elements. The integers to the right of the entity type name in the iset widgets in Figure 5 indicate the order of the elements in the tuple entities (these labels will be hidden for end users). The `node_1` of any of the 499 tuple entities is therefore a `Contact`, and the `node_2` is an `Email` message. The DQ filters operate on derived attributes like `node_1` ∘ `Company` (where ∘ denotes composition). Using derived attributes, the DQ implementation doesn't have to know anything about navigation isets. Tuple entities are also essential in record-level visualizations that encode properties of multiple entities, such as a scatter plot of email message date versus sender's company.

The two "node" result isets are dynamically linked via the join iset, because filtering on the [composed] attributes of one changes the subset of join records, which in turn affects the counts and histograms for both result node isets. Due to projection, the histograms represent distributions of base domain entities (messages and contacts) rather than abstract join records. As a result, users can explore the relationships among multiple entities while thinking only of those entities. Linked histograms for a single data table go back at least to DataDesk [Velleman 1993].

While the dark conditional histograms are projections from the join result, the light gray unconditional histogram must be computed on the Email or Contacts universe isets rather than the join universe due to the possibility of lossy joins.

Our thesis is that seeing the cardinalities and histograms for both universe and result isets, as well as any other intermediate results, is more useful than the results of a single SQL query. For the user, the two universe isets allow drag and drop updating of the query universes. The result isets present results in terms of the original domain entity types. Their cardinality is the count of something meaningful, contacts or messages, rather than a count of contact/message tuples. We are making the strong assertion that neither tuple counts, nor any other attribute of join views, are of any value to the user. After all, using English to talk about properties of combinations of things in general is awkward. When a kind of tuple is worth talking about, it is given a name. For instance, an insurance company might use the term "visit" to refer to an event involving doctor and patient entities. The only time we have felt a need for this kind of reification has been analyzing user activity logs, where we want to understand the higher level tasks that users accomplish with sequences of commands. In this odd case, the database schema is known not to correspond to appropriate intuitive concepts. A reasonable extension to Visage might be to annotate relationships that are reified as nouns, and to show the navigation iset pair in this case. All previous database query systems supported interaction with multi-entity query results by forcing users to work with these abstract constructs. In order to find counts of domain entities, users would have to write an SQL expression using DISTINCT corresponding to each iset (see Figure 6).

```
SELECT COUNT (DISTINCT recipients.domain), COUNT(DISTINCT Contacts.ID)
   FROM Contacts, recipients
         WHERE Contacts.ID = recipients.range AND Contacts.Company = '';
```

Figure 6 Oracle SQL query that calculates the cardinality of the two result isets visible in Figure 4.

2.4. Partition

It is useful to visualize isets of isets, analogous to SQL `GROUP BY` queries. Figure 7 shows how VQE represents a query over messages grouped by their `senderName`. The arrow labeled "partition" is dashed to indicate that it is not serving the normal role in linking the iset widgets. It represents a relationship between the isets themselves, rather than one among their members. The stack icon inside partition iset widgets increases the salience of the distinction. Note that there are two groups of attributes visualized in partition iset widgets. 1) The lower group (containing `max_ReceivedDate`, `SQL`, etc) fills the normal role of filtering members of the universe iset. Since the members are themselves isets, most of the attributes are derived, like `max_ReceivedDate`, which is the name given to a derived attribute defined by the expression `MAX (ReceivedDate)`. Visage creates new entity types, such as "`Email_iset`," and their attributes as needed. 2) To the right of the stack icon is a set of attributes to `group by`. Initially this set is empty, and the partition contains only

one member. In Figure 7, the `SenderName` attribute has been copy-dragged from the Email iset widget to the top region of the partition iset widget. It is labeled "`by SenderName`" to emphasize that it is a parameter of the iset, rather than an attribute of the members. If additional attributes from the Email or Contacts iset widget are dropped on the partition iset widget, the partition will be refined so that its members represent the cross product of all the attribute values. This is equivalent to an OLAP cross-tab query, but without any commitment to a particular visualization. We could build a cross-tab interface in Visage and drag the result `mail_iset` iset to it. The advantage is that VQE navigation, filtering, and drag and drop operations can be interleaved with conventional slice and dice.

VQE allows drag and drop partitioning by attributes from any query node. For an attribute like `Company`, which does not apply to email directly, this is implemented by first defining a new attribute of `Email` that reflects the company of the sender. Then this new attribute is added to the list of attributes to `group by` in the partition iset widget (e.g. "`by sender_Company`"). VQE also defines auxiliary variables in order to partition by quantitative attributes like `ReceivedDate`. For quantitative attributes, users may adjust the number and widths of bins to `group by` with a menu or with direct manipulation.

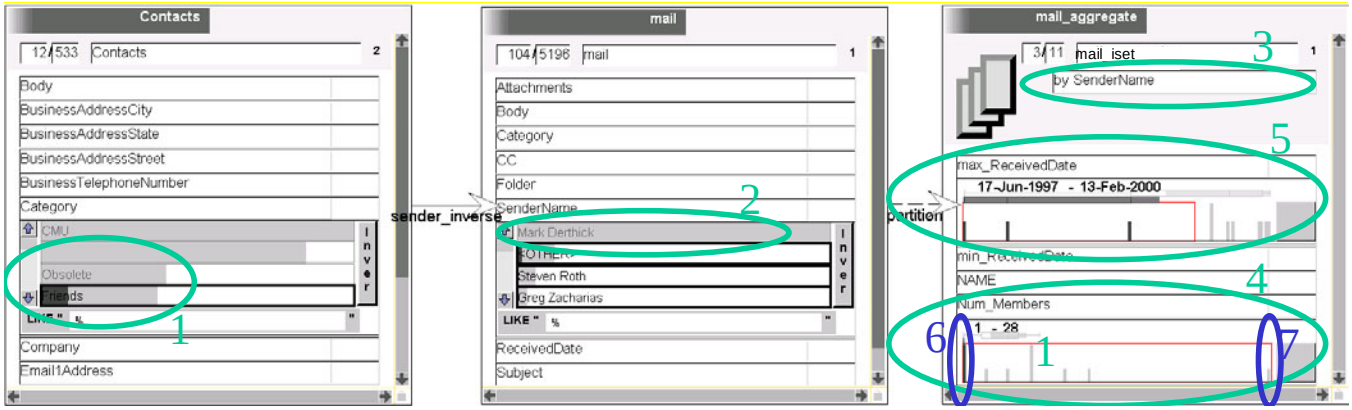

Figure 7 The rightmost iset widget partitions the 104 email messages from friends (see oval 1) other than md (see oval 2) by `SenderName` (oval 3) and shows the number of messages (oval 4) and the date of the most recent message (oval 5) from each. The `max_ReceivedDate` slider selects the [three] friends who md hasn't heard from in two years in dark gray. On the `Num_members` histogram, all and only the singleton collections are selected (oval 6), which indicates that md only received a single message from these "old" friends. md has not received more than 28 messages from any friend (oval 7).

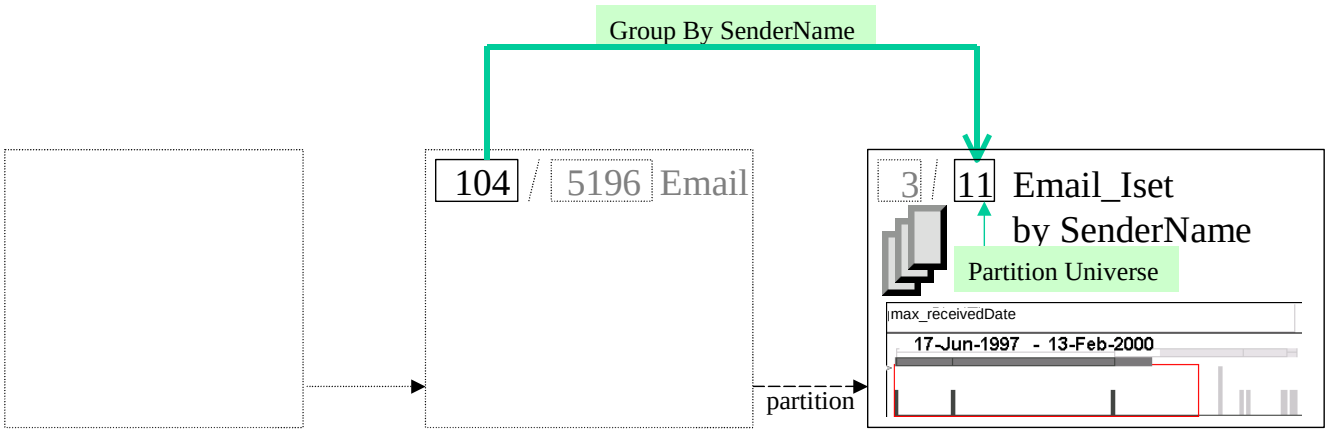

Figure 8 Data flow among isets for partitioning.

The constraints connecting the two isets are rather different than in the join case, as shown in Figure 8. A partition universe iset is derived from a source result iset, so the example partition must be recomputed for every `Email` or `Contacts` DQ change. This results in unacceptably slow DQ feedback on large datasets. One way to make it fast, at least when the number of collections is small, would be to define a new email variable for collection identity and compute iset properties using the same kind of index used for computing histograms (see Section 3.1).

Iset pairs make it easy to compare subsets to their universe. It is also useful to compare multiple subsets with each other and with their common universe. Tufte [1983] calls this technique "small multiples." In Visage, visualizations of partition members designed by the SAGE expert system [Roth and Mattis 1990] provide some of this functionality. The properties of each instance of `Email_iset` can be shown using charts or maps. Unfortunately, SAGE doesn't support nested visualizations, which systems like VIQING use to create small multiples [Olston et al. 1998]. We could write code linked to a special-purpose interface gesture that creates multiple SAGE pictures. From Figure 7 (middle), for example, dragging a copy of the `senderName` attribute to a SAGE visualization would indicate the intent to duplicate the visualization for each value of `senderName`, and populate it with messages having that `senderName`. Thus, the user wouldn't even have to go through the intermediate step of creating the partition iset widget.

2.5. Set Union, Difference, and Intersection

Dragging a representation of an iset onto an iset pair widget of the same entity type performs set union. Dragging one or more entities of that type onto an iset pair widget has the same effect. Dragging a result iset representation out of an iset pair widget performs set difference. There is currently no way in VQE to perform intersection or set difference in general, although the need is clear. These features could be implemented with additional drop targets labeled '–' (for set difference) and '∩' (for set intersection).

2.6. Copy and Clone

Visage offers four dragging modes that control both visual objects and the data objects they refer to. These modes are most easily explained in relation to those of Microsoft Windows, even though the tasks Windows support are somewhat different. Windows offers three dragging modes: move, copy, and create shortcut. In a move, the underlying data object is removed from its current location and added in another location. This is the default mode in both Windows and Visage. Creating a shortcut provides access to the same data object from two locations. Copy provides access to identical data objects from two locations.

Visage's dragging modes are slightly different because Visage entities are not so closely tied to locations as files are to folders in Windows. In Windows, the only way to refer to or navigate to a file is by folder. Dragging a file to the trash removes it from its folder, so there is no way to refer to it any longer. It is effectively deleted. In Visage, an entity may be accessed by navigating along any relationship. Thus dragging (moving) an entity out of a visualization and into the trash does not delete the object; it just deletes it from the iset being displayed in that visualization. Because entities exist independent of any visualization, it makes sense to break down the Windows dragging modes into two distinctions: first is whether the data content is removed from the source location (which we call move vs. copy), and second is whether the destination provides access to the original data object or to a distinct object with the same attributes (which we call share vs. clone). Then the Windows modes can be characterized as follows:

Windows Mode	move/copy	share/clone
Move	move	share
Shortcut	copy	share
Copy	copy	clone
N/A	move	clone

The fourth mode, move+clone, is not useful in Windows, because if a file is removed from a folder without being added to another folder, it can no longer be accessed. In Visage, the shift key adds the copy property to a drag and the control key adds the clone property.

In VQE, dragging the result iset label out of an iset widget has a slightly different meaning than for most visualizations. In the move modes, the members of the result iset are removed from the universe iset. For each of the four drag modes, Figure 9 shows the effect of dragging the result iset of Figure 3 rightward and dropping it on a VQE to form a

new iset widget. For moves, the original iset pair will end up with 0 of 105 contacts selected (A&B). If the visual object is copied instead, the original iset is unchanged and will still contain 13 selected contacts of 118 (C&D). If the iset is *cloned* (B&D) (which happens before the copy or move), the new iset will have cardinality 13. If the isets *share* (indicated by brushing in A & C) the universe iset of the new iset widget will track the result iset of the original iset widget. If the bar representing the empty company string is selected again in the original iset widget, the cardinality of the two *sharing* cases will increase by 105, either to 105 (A) or 118 (C).

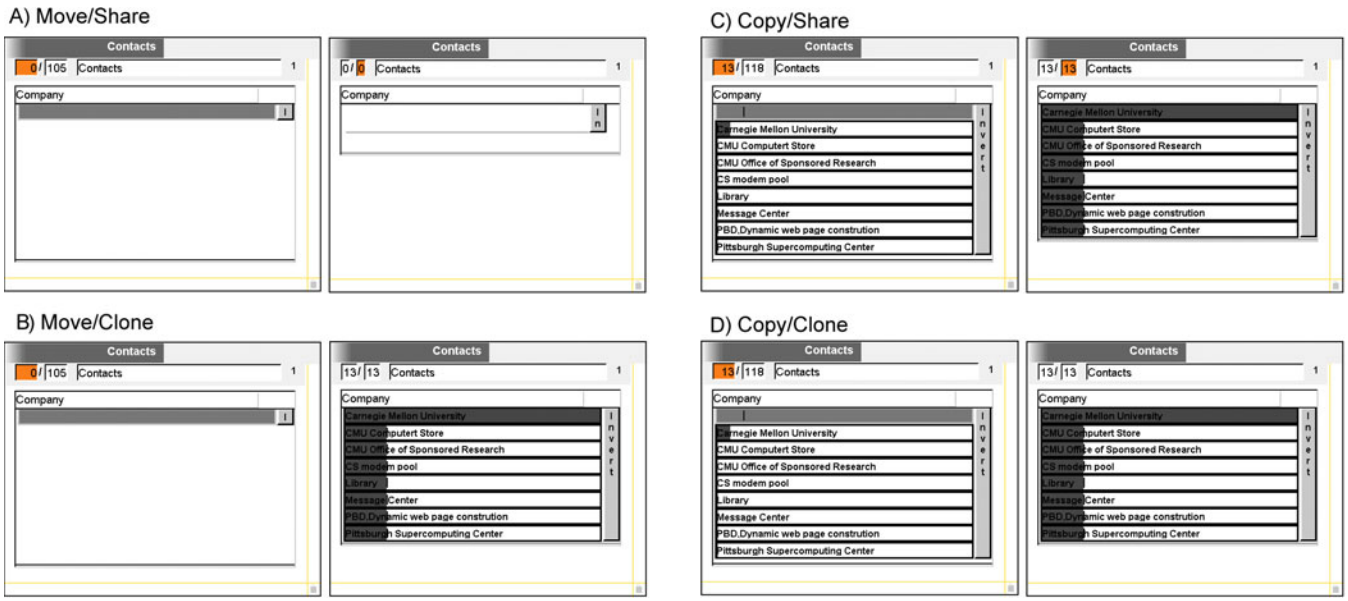

Figure 9 The four dragging modes for creating new iset widgets (right columns) from existing ones (left columns).

The four combinations are useful in different situations. From Figure 3, (C) and (D) make the most sense, because they retain the focus on the 13 contacts that have company information, allowing further exploration of this subset. Case (D) is useful to start a new line of inquiry, while (C) is useful if further filtering in the original iset is expected. An alternate strategy is to remove unwanted contacts, rather than filter them. Using the "invert" button on the company DQ widget, the user can select the subset that does not have a company and then use case (A) to more permanently remove these uninteresting contacts in the left iset. In this case the removed iset would more likely be dropped in the trash than back onto the VQE frame. This method leaves less clutter on the screen, but by the same token removes a visual reminder of the analysis sequence. Case (B) is useful if filtering has been used to divide the universe iset into two interesting subsets that the user wants to analyze separately.

Sequences of filter and drag operations can be used to get the effect of database Boolean and set operations. On the plus side, this avoids the need to use syntax, which is hard for many users to get right [Ahlberg et al. 1992]. Unfortunately

the direct manipulation approach lacks a declarative visual representation of the derivation of the final iset state. In ongoing research we are exploring ways to make it easier for users to remember the intent behind isets or other intermediate results by visualizing the operations that produced them, providing undo/redo sliders to animate the history, and generating explanations [Derthick and Roth 2001a].

Sharing (A&C) is a good way to focus on details, because it re-scales the histograms in the right iset widget to fit the subset of entities selected in the left iset widget. Figure 10 shows case (C) except that all companies were reselected, and all categories except Friends were deselected. The `company` DQ widget has been scrolled to show that the "USC Information Sciences Institute" bar is only one pixel long. It is nearly indistinguishable from its neighbors, such as "Kraynick's bike shop," where md has no known friends. In the focused iset widget on the right, there are two advantages for exploring the small subset picked out on the left: First, the histogram bars are rescaled, so it is evident that md has [told Outlook about] more friends at USC than at Microsoft Research. Second, the top 15 values to display are chosen for this subset. Thus values that are so rare that they are grouped under <OTHER> on the left, such as the University of Pittsburgh, show up individually on the right. Further, the elimination of companies where md has no friends brings those where he does in proximity for easier comparison. Since this technique is designed to look at small subsets, Visage has no code to bypass the database in order to optimize the interaction. After every DQ action on the left iset widget, the SQL for the right iset widget is recomputed, the database is queried, and the indexes that support DQ in the right iset widget are rebuilt.

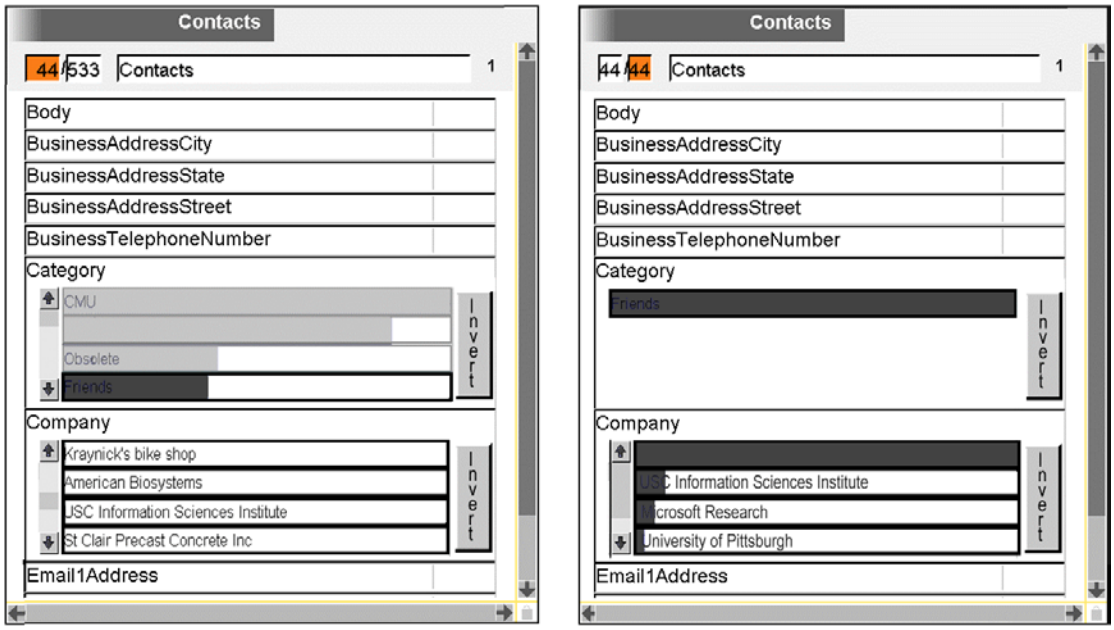

Figure 10 Using sharing to zoom into a small subset and rescale histograms.

Previous systems have not considered all these situations in which the analyst may issue multiple related queries. The principled set of four dragging modes allows easy modification of queries in task-appropriate ways. Layout clarifies the relationships between universe and result isets, making it easier to keep track of this common sequence of the analysis process.

2.7. Brushing

Brushing was first proposed by Becker and Cleveland [1987]. Selecting visual objects highlights all visual objects representing the same entity. Visage offers a menu of multiple brushing colors, and implements brushing with an explicit attribute. Selecting a visual object sets the attribute value of the entity it represents to an integer representing the color. Then all visual objects representing the entity update their color. As discussed below, this allows isets to be defined by brushing. It also lets end users customize brushing feedback in a particular visualization by mapping the `brushing` attribute to position, size, or shape.

3. Implementation and Larger Examples

3.1. Visage

The incremental semantics discussed in this paper have been implemented on top of the Visage data exploration and visualization system, developed by Carnegie Mellon University and Maya Design Group [Roth et al. 1997]. Since the motivation for the new semantics is interactive exploration, system issues like performance are relevant to their evaluation. This section outlines the architectural relationship between the visualization system and the data repository (see Figure 11).

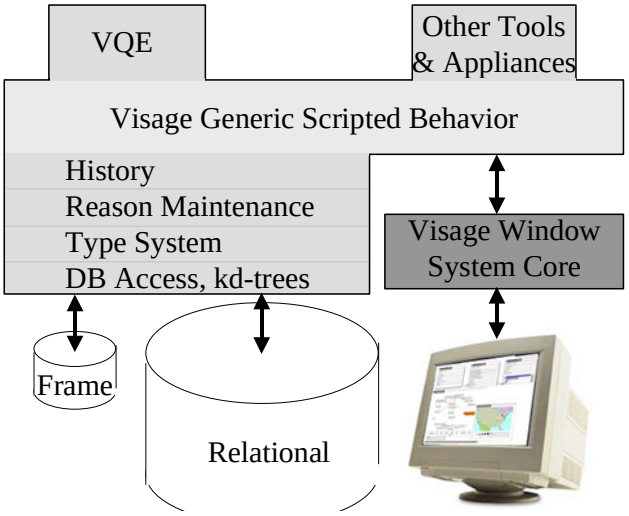

Figure 11 Visage Architecture. Most behavior is scripted, including layered data semantics and tools like VQE. Core window functionality and rendering is implemented in C++.

A scripted database access layer transforms queries over isets into SQL and sends them to the relational database. If Visage needs to display the iset members individually, information from the relational database is used to create explicit objects in the frame repository. The type system specifies the domain and range entity type of relationships, and supertypes of entity types. If the data do not include type information, it is assumed that domains, ranges, and entities are simply Things.

When a database is registered with Visage, it extracts the schema that by convention is encoded in two tables, `entitytypes` and `relationships`. There must be a database table for every entity type and either a table or a foreign key for every relationship. At registration time, VQE performs queries to find functional relationships, which it uses to improve query efficiency. To work with an existing database that doesn't conform to an ER schema, it is usually a simple matter to define views that present VQE with one. Visage issues SQL queries and updates to maintain synchrony between its object-oriented frame data structures and the relational database. In particular, isets lazily compute their extension as needed by visualizations.

Because relational databases don't efficiently support histogram queries, special-purpose indexes are built to allow fast DQ interaction with navigationally related isets. These *kd*-forest data structures support 100ms feedback for up to a million records on a 400MHz, 384MB Windows PC [Derthick et al. 1999].

Visage provides for derived attributes using either a declarative language or by attaching procedures. All inference is lazy and cached, and a reason maintenance algorithm decaches invalidated attribute values and sends notifications to affected visual objects. All recomputation is triggered directly or indirectly by the need to display a specific attribute value.

3.2. Filtration

Registration of a database with Visage adds isets representing the records in each of its entity type tables onto the workspace. In Figure 12, the user has dragged a copy of the `Email` iset, whose definition is "`SELECT * FROM Email`", into a VQE frame. When Visage's repository is asked for an attribute of the iset (e.g. `COUNT`) the SQL definition is used to pass the query on to the database.

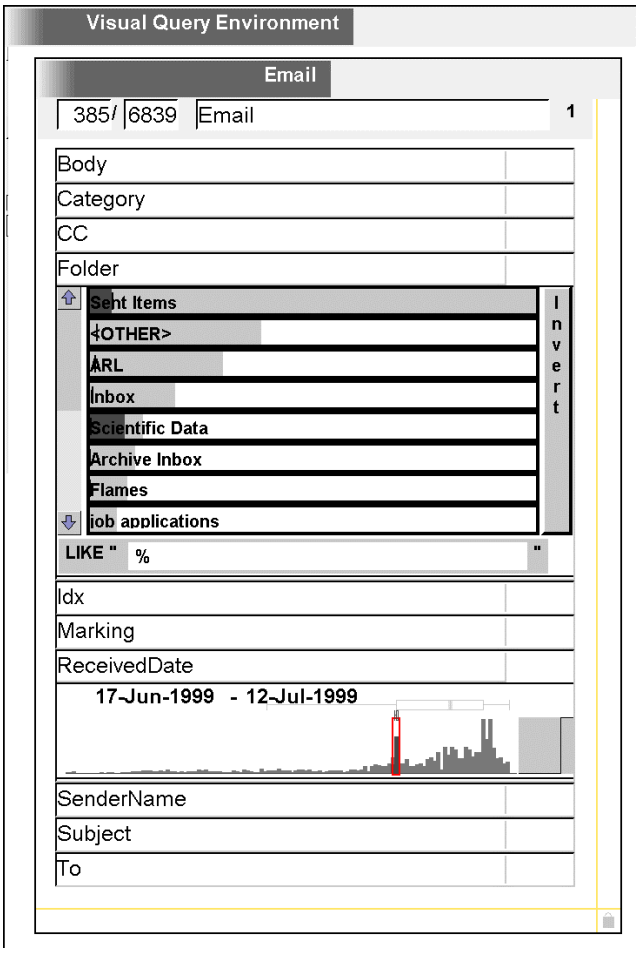

Figure 12 Using DQ to filter isets.

In the figure, DQ widgets have been dropped on the `Folder` and `ReceivedDate` attributes. There is a peak in the `ReceivedDate` distribution from about 17 June to 12 July 1999, and this range has been selected. From the `Folder` distribution of the result iset, it is apparent that almost all the incoming messages during this period were put in the Scientific Data folder. This folder was created for a proposal that was due 8 July. The line chart to the right of the ReceivedDate histogram is a ROC (Receiver Operating Characteristic) curve [Provost and Fawcett 2001], which serves a similar purpose to the hash marks on histograms for nominal attributes. An ROC curve monotonically increases from the lower left corner to the upper right corner. If the attribute is independent of the conditions imposed on the result iset, the curve will be a straight line (45 degree diagonal). Here the result set condition is entirely defined by restrictions on ReceivedDate, so the ROC values are all at the extreme high or low y values, and far from the diagonal. The light and dark histograms together contain enough information to calculate the ROC curve.

The conditional count and histograms in the iset widget must be recalculated after every DQ click that toggles a histogram bar on or off for nominal attributes and after every mouse move for quantitative attributes. Doing this by

updating the SQL and querying the database is too slow when the iset has 10^5 or 10^6 records. Each DQ filter sets up its own index and calls the indexing code directly to eagerly update the count and histograms [Derthick et al. 1999]. When change notifications are sent to the numeric label and the histogram polygon, the data they need for redisplay is therefore already cached, even though the SQL for calculating it from a database query is not. Thus the fallback lazy computation incurs no overhead during DQ. The index must fit into RAM, which limits VQE to interactive exploration of a few hundred thousand data points. The indexes must be rebuilt when new DQ widgets are added, when the query structure changes, or when the universe iset's membership changes.

3.3. Filtration Chains

Visualization of iset pairs allows part/whole comparison and access to intermediate results. These features can be just as useful for chains of subsets longer than two. Some of the reasons for chaining are:

1) To show more intermediate results.
2) If the subset's cardinality is only a fraction of the universe's, that fact will be apparent in the histogram, but its distribution may be almost invisible at the scale of the universe. Making it the universe for another iset widget will generate an appropriately scaled histogram.
3) Filtration chains can also be used to scope the exploration. For instance, in order to account for her time over the last month, an employee may first restrict an email iset to that time period, and then do further filtering based on folder or recipient company.
4) In order to investigate multiple alternative scenarios, a separate universe can be constructed for each by creating an iset widget from a copy of the result iset.
5) If the join cardinality corresponding to a query graph is much larger than the individual universe isets, breaking the graph into multiple subgraphs can reduce computation connected by shared isets. The worst-case universe size of a navigation iset is $O(m^n)$, where m is the maximum cardinality of a base-level universe iset, and n is the number of nodes in the graph. A 3-node graph of 10,000-element isets is $O(10{,}000^3) = O(10^{12})$, while two 2-node graphs require two indexes that are each $O(10{,}000^2) = O(10^8)$.[1] However the navigationally downstream index must be rebuilt each time the upstream iset is filtered.

Cases 1, 2, 3, and 5 involve sharing the result iset, in which the new universe iset's visual representation denotes the same entity as the original result iset. Case 4 involves cloning, in which the new universe is a copy of the original result.

We built a digital video library application that uses sharing to build a long chain for reasons 1, 2, and 5. It is interesting to note that it was entirely built interactively, using VQE to perform the data manipulation and SAGE to design the visualizations. Figure 13 shows its use on a dataset from the Video TREC 2002 evaluation [Smeaton et al. 2002]. On the top left are widgets for querying the 24,263 shots (termed "`shotbreaks`" in the figure). The `transcript_relevance` attribute assigns an estimated relevance to each shot for the text query "catsup ketchup manufacture tomato heinz," and the DQ widget is set to show those with a relevance just greater than 0 (on a scale of 100). It also shows the cardinality of the result. Keyframes for the shots that will fit, sorted by `transcript_relevance`, are displayed in the visualization at the top right. In video, we have found that the transcript is often not temporally aligned with the images it refers to. In addition, if the keyframe shows an image related to the query, another image from the same video segment is often an even better match. Therefore, the interface is designed to let users select (brush) a subset of the images that seem promising, and it then displays all the shots from the segments including those shots. Doing this efficiently and displaying helpful intermediate results required a tree of 13 connected isets.

[1] This assumes that the graph is singly connected (i.e. a tree), which is almost always the case with query graphs. If all three nodes are connected, the example requires three 2-node indexes.

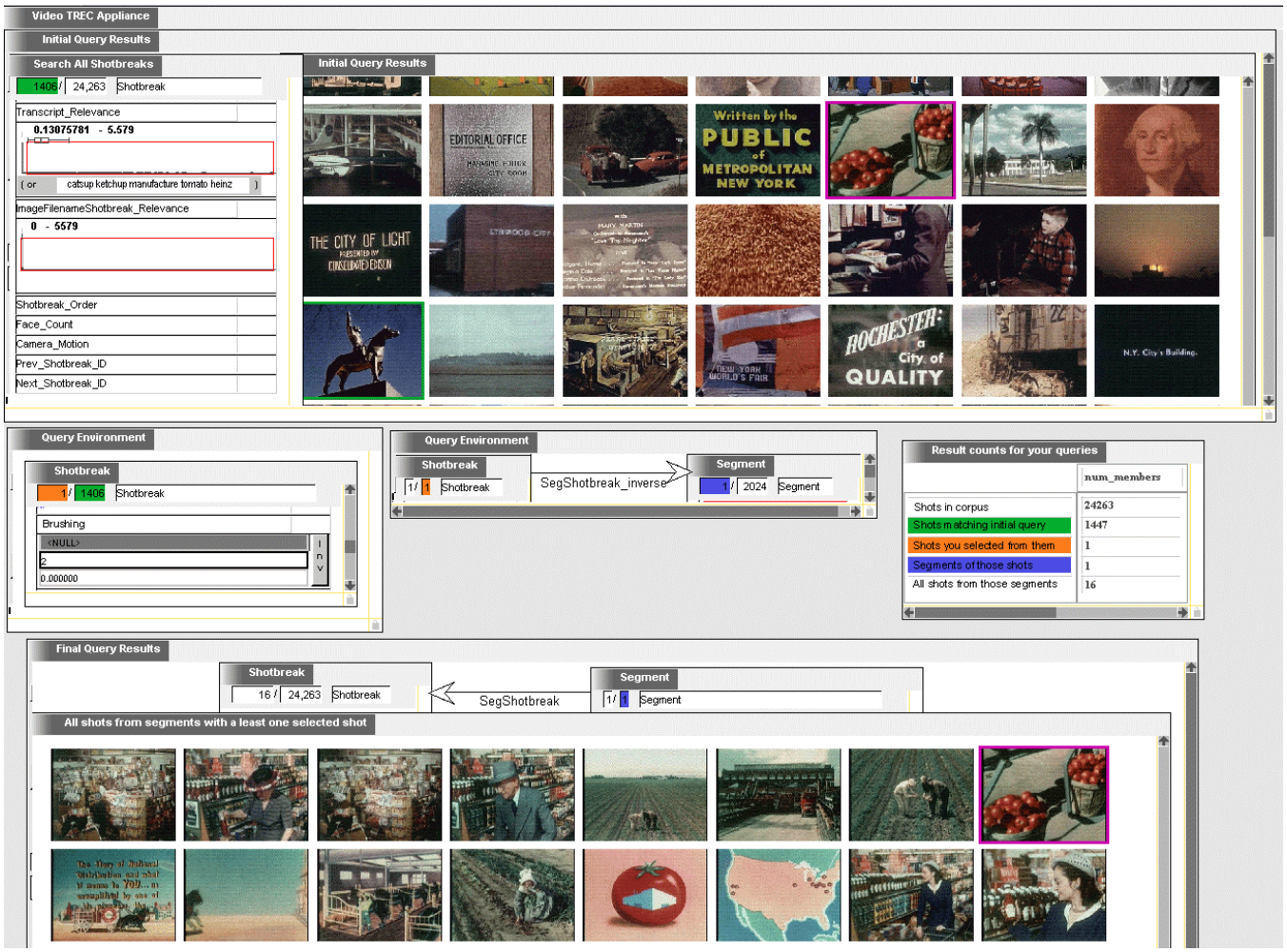

Figure 13 Example query and selection using the TREC appliance. The interface has been rearranged to make the data manipulation explicit. Normally, neither of the two middle VQEs are visible, nor are the iset widgets in the bottom VQE. The cardinalities are displayed in the table on the right in the middle instead.

In the figure, each of the three shared isets that connect parts of the tree has been brushed with a different color to show the co-reference relationships. The green iset connects the text query to the query based on brushing. Each brush color is represented internally by a positive integer value for the attribute "brushing". Zero (or null) means that the object is not brushed. In the leftmost VQE in the middle, zero and null have been deselected, so the result iset includes only brushed shots. (The only positive integer that shows up in the histogram is 2, because no other colors are being used to brush shots.) Here the user has brushed the image of tomatoes in magenta (color 2, shown as a magenta rectangular outline), so this result iset has a cardinality of 1. Sharing was used for reason 1 above, to display to the user the intermediate result that the text query returned 200 shots. Both constraints could have been imposed on a single iset widget, but then only the cardinality of shots satisfying both (1) would be apparent. This orange result iset is then shared for reason 2, so that it is faster to build an index for navigating from shots to segments, as the number of selected shots is expected to be much smaller than the number

returned by the text query. The VQE to the right then navigates from the single shot to its single segment. The blue iset is used to navigate from the segment(s) back to all the shots they contain, for reason 5.

In writing up this example it became apparent that there is no need for the blue sharing in this application, both because the number of brushed shots is so small and because the ratio of shots to segments, 12, is not very large. In general, navigating from *O(n_shots)* shots to segments and then back to shots will generate $O(n\_segments * (n\_shots / n\_segments)^2)$ = $O(n\_shots^2 / n\_segments)$. Each segment will join with each of its approximately *n_shots / n_segments* shots in both of the shot isets in all possible pairs. Using the blue share requires two indices, both of size *O(n_shots)*. (The combinatorics here are better than those given in reason 5, because `SegShotbreak` is a many-to-one relationship, rather than the worst case of many-to-many.) The downstream query's index does not have to be rebuilt any more often than the upstream query, because the upstream iset widget doesn't have any DQ widgets.

3.4. Joins and Projection

Maintaining indexes to support fast DQ is more complicated in the case of multi-entity queries, because they must support projection in addition to counting. An ordinary *kd*-tree [Bentley 1979] node maintains a single count of all the records it represents. At query time, the shallowest set of nodes that exactly cover the query is determined. Each of these nodes represents a disjoint set of records, so the cardinality of the record set matching the query can be found by adding the counts. Here we are counting the number of distinct values of the `node_1 ∘ ID` attribute for histograms of contacts and of the `node_2 ∘ ID` attribute for messages. Because `recipients` is a many-to-many relationship, the same ID values can occur in multiple records. For instance, if a message is sent to two contacts, its ID will occur twice; therefore, nodes must encode the set of IDs they represent so that the number of distinct values can be recovered from the covering node set. An exact solution to this problem requires each node to keep a bit-vector as long as the number of distinct ID's, *n*, which can be hundreds of thousands. Fortunately there is an approximate algorithm for counting unique values that requires only *O(*log *n)* space at each node [Flajolet and Martin 1983]. Thus the multi-entity DQ algorithm is less efficient by a constant factor than in the single entity case, and only provides approximate counts and distributions. Currently, VQE obtains 5% precision with 1024 bits stored at each tree node for each attribute (independent of aggregate cardinality) [Derthick et al. 1999]. This approximation does not noticeably distort the histogram distributions. Since the counts for the isets are displayed as precise integers, however, the inaccuracy causes more of a problem. It is disconcerting to see an iset widget summary line like "`5200/5000 messages`." We could turn off updating the count based on the *kd*-tree index and rely on the SQL definition; however, on balance it seems that fast response time is more important than precision. (Users can force

recomputation of the precise value by copy-dragging the iset to a VQE frame, which issues a real database query in order to build an index.) There are two special cases where results are always exact. If VQE knows that a relationship is many-to-one, indexes for DQ filters associated with the "many" node use ordinary *kd*-trees. If there are fewer than 1024 members of a universe iset, the bit vector representation is used.

Ioannidis [1996] also applies DQ to data that require multiple tables by creating multiple queries. Rather than projecting two result isets from a join query, he suggests nesting one query within the other. Thus, it is possible to filter the results of one entity type based on attributes of another, but the relationship is not symmetric. It is also possible to recursively define each filter as nested inside the other in order to provide symmetry. Neither nesting scheme can support record-level visualizations that encode attributes of multiple entity types.

3.5. Navigation

Combining navigation with universe modification allows answering the question "Who has md phoned but never emailed?" (Figure 15), which is an awkward problem for previous systems (see Section 5.3). This question arose while making an invitation list for a party. A previous email invitation list was available, but there was no such list for friends without email. Starting from the set of phone calls and navigating to the `Contacts` called yields a universe iset of 66 people who md phoned. Navigating from there to the messages those people received and selecting only those messages in which `senderName` is Mark Derthick leaves in the `contacts` result iset 48 people who md phoned and emailed. This state is shown in Figure 15. Dragging out these 48 will perform set-difference and leave the desired set of 66-48=18 people as the universe iset of the `contacts` iset widget. The total number of mouse gestures required is nine. A video illustrating this example is available at http://www-2.cs.cmu.edu/~sage/animations/PhoneNoEmail/ExpressiveQueries.smil.

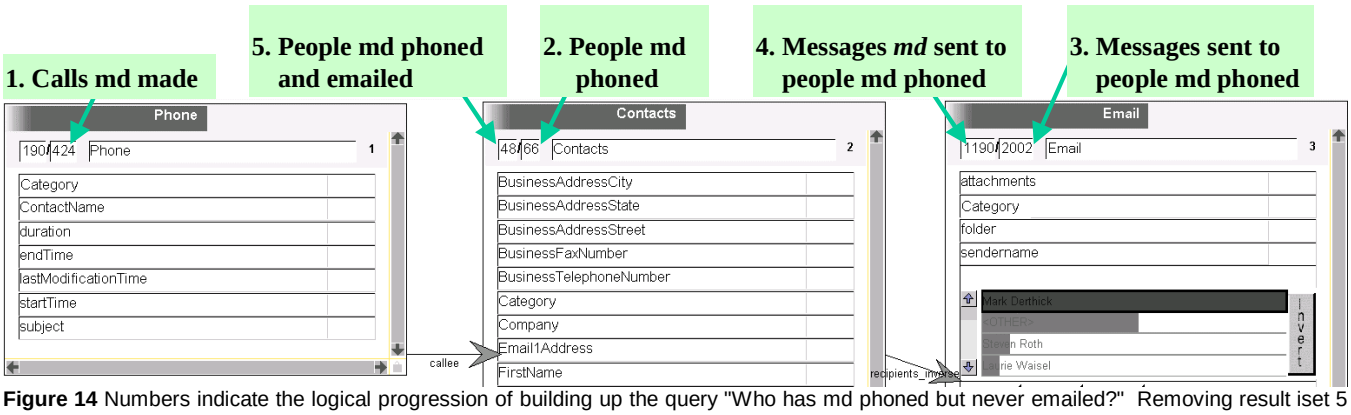

Figure 14 Numbers indicate the logical progression of building up the query "Who has md phoned but never emailed?" Removing result iset 5 from universe iset 2 will leave the answer in universe iset 2.

Notice that we sidestepped the difficult problem of directly defining the set of people md did not email. The ability to break out of the declarative query framework and perform sequential set additions and removals often simplifies data exploration. The downside is that without an explicit representation users must remember what the set of 18 people means. In contrast, the SQL for answering this query would explicitly contain "`...WHERE NOT EXISTS (SELECT...`" In fact the `Contacts` universe iset has an `SQL` attribute that includes this restriction. The full SQL for this iset contains 37 tokens. Even a human-generated SQL expression that directly answers the question is cumbersome, requiring 25 tokens (Figure 15). Showing anything like this to the user is not likely to be helpful. A visualization of the operations performed would offer a simpler way to explain the intent of the final state of the result contacts iset (see Section 6).

```
SELECT DISTINCT fullName FROM callee, Contacts
      WHERE Contacts.ID = callee.range
      AND NOT EXISTS
            (SELECT * FROM recipients, Email
                  WHERE Email.id = recipients.domain
                  AND Contacts.ID = recipients.range
AND Email.senderName = 'Mark Derthick')
```

Figure 15 SQL query for "who has md phoned but never emailed?"

For operations other than drag-and-drop (i.e. operations that don't directly change a universe iset – navigation, filtration, projection, partition), the paired isets of the iset widgets and the navigation arrows linking them provide a visual trace of an analysis sequence. Previous interactive systems do not have first class entities representing query linking. Hence, they are handicapped in supporting the user's recall and understanding of the intent behind queries. Dataflow languages represent inference chains, but their batch orientation is incompatible with VQE's continuous feedback exploration. Further, their procedural semantics are less flexible than VQE's declarative semantics, which supports bi-directional propagation.

3.6. Exploration in the large

VQE navigation paths are a visual representation of an analysis process. SAGE visualizations are easily recognized analysis landmarks. Dragging out all members of all the isets leaves the structure in place, allowing it to serve as a template for doing the same analysis on other data. Annotations can be added, and the layout can be edited to maximize usability. In some cases visualizing the query graph may just add confusing clutter and would be cropped during this process. For reuse, dropping new data on any VQE node, or any SAGE picture inside VQE, will propagate along navigation paths in both directions to re-create the analysis. An analyst or an organization can build up a library of these custom "appliances" and

store them as thumbnails in a Visage tool palette [Derthick and Roth 2001b]. Local experts can create appliances to be used by less experienced users, as commonly happens with spreadsheets. The same process can be used to document an analysis for formal presentation in Visage slide show frames [Roth et al. 1997].

4. Formal Semantics and Expressive Power

4.1. Operator Definitions

Each operation is defined below by mapping it to SQL. Below are the typographical conventions used and the mapping to SQL for each of the operations:

Italics denote expressions that are evaluated; `UPPER CASE` variables represent literal SQL strings.

domain(edge) and *range*(edge) return the nodes that an edge connects.

domain(DQ) returns the node the widget is inside visually.

attribute(DQ) returns the attribute the widget filters.

selectedValues(DQ) *(unselectedValues*(DQ)*)* returns the set of selected (unselected) values of a nominal widget.

min(DQ) and *max*(DQ) return the upper and lower slider settings for quantitative DQ widgets.

attributes(node) returns the set of attributes for the node's datatype.

SQL(iset) returns the definition.

table(entity type) and *table*(relationship) returns the name of the entity type or relationship table in the database.

parent(filter node) returns the universe iset for this result iset.

tupleElements (query graph) returns the table mapping a tuple of member IDs, one for each query node, to the corresponding tuple entity ID.

A **filter** query for *node* with *q* quantitative DQ widgets, *s* nominal DQ widgets where <OTHER> is selected, and *d* DQ widgets where <OTHER> is not selected generates the SQL:

```
SELECT * FROM (SQL(parent(node)))
    WHERE domain(DQj).attribute(DQj) > min(DQj) AND
          domain(DQj).attribute(DQj) <= max(DQj) |1 ≤ i ≤ q AND
          domain(DQk).attribute(DQk) NOT IN unselectedValues(DQk) |1 ≤ k ≤ s AND
          domain(DQp).attribute(DQp) IN selectedValues(DQp) |1 ≤ v ≤ d
```

A **navigate** operation creates a new iset of type *t* from an existing iset *n* and relationship *r* with the definition:

```
SELECT * FROM table(t) WHERE EXISTS
    (SELECT * FROM SQL(n) e, table(r) r, table(t) t
          WHERE e.ID = r.domain AND t.ID = r.range)
```

A **join** query for a query graph that has *n* nodes and *v* edges has the form:

```
SELECT * FROM (SQL(node_i))  node_l, |_{1 ≤ l ≤ n}
                              edge_m edge_m, |_{1 ≤ m ≤ v}
                              tupleElements(queryGraph) tuple
   WHERE
      edge_i.domain = domain(edge_i).ID AND
         edge_i.range = range(edge_i). ID  AND |_{1 ≤ i ≤ v}
     node_l.ID = tuple.ID_l (+) AND |_{l=1 to n}
```

A **project** query extracts from a join query the attributes for the entity type of one node. For node *l*, the intent of the SQL is shown below, although to avoid verifying distinctness every attribute we first collect the distinct key field values with one SELECT, and then use a self-join to collect the other attributes.

```
SELECT DISTINCT node_l.a |_{a ∈ attributes(nodel)} FROM SQL(joinIset)
```

A **partition** query regroups members of one *node*. For each quantitative *attribute* grouped into *b* buckets, VQE defines an auxiliary attribute:

```
UPDATE entityType(node) ADD
     SELECT DECODE (node.attribute <= max(bucket_{i, attribute}), i, |_{1 ≤ i ≤ b,
                                                ELSE b+1)   AUX_DQ
```

Then the partition SQL for *node* is:

```
SELECT * FROM (SQL(nodePartitioned(node)))
      GROUP BY a, |_{a ∈ partitionAttributes(node)}
```

Set operations can be implemented with SQL UNION, INTERSECT, and MINUS, or in special cases by AND, OR, and NOT.

Copy and **clone** do not change a query's SQL representation.

Brushing simply updates the brush color attribute:

UPDATE *entityType* SET brushColor = DECODE(brushColor=*currentColor*, 0, ELSE *currentColor*) WHERE ID = *entityID*. Brushing a tuple entity is implemented as brushing each of its elements.

Our goal is to support extensive exploration without the need to exit VQE, modify the dataset, and restart; therefore, VQE also supports creation of **derived attributes and relationships** by drag-and-drop on the query graph. For instance, a

new Contact attribute for the number of messages received can be defined by dragging a copy of the `ReceivedDate` attribute from the `Email` iset widget to the `Contacts` iset widget in Figure 4. Since `recipients` is many-to-many, the user must choose from a menu of aggregation functions, and in this case chooses `COUNT`. If the definition requires operators other than navigation and partition, such as "`+`" or "`If…Then`", users must type them using SQL syntax.

One danger of incremental queries is that the meaning of a query can become dependent on its derivation process, and no longer recoverable from a declarative representation. In the current version of VQE, the meaning of a query can be read from its visualization except for the universe isets. This is similar to using named views in SQL, where you have to understand the views in order to understand the query. One common case is that the first universe iset created contains all instances of an entity type, and the remaining universe isets are created by navigation. Assuming this condition makes the query interpretation unambiguous. When this condition fails to hold, additional information could be added, as suggested in Section 6.

4.2. Expressive Power

This subsection shows how to map in the other direction, from SQL to the ten operations plus derived attributes. Our language is less expressive than SQL, so some constructs require procedural drag-and-drop set operations, and some can't be expressed at all. First, the VQE language only supports `SELECT` statements. `UPDATE`s, `INSERT`s, and `DELETE`s can be specified on record-level visualizations. Selecting the `TOP` *n* results is also done with a widget on a record-level visualization. Selecting an expression can only be accomplished by first defining an attribute with the expression. `INTO` is not necessary since isets are first class and persist. `FROM` arguments become isets that can correspond either to base entity tables or derived views. Navigation operations in effect perform both an `INNER JOIN` and a `LEFT OUTER JOIN`. The two result isets represent the inner join, while the originating universe iset still includes unmatched objects. The destination universe iset can be converted to a `RIGHT OUTER JOIN` by dragging the entity type label onto its iset widget, which will add all the entities of that type. Only equi-joins on key attributes are supported. `GROUP BY` is handled with partition, and requires defining new attributes for grouping by an expression. `HAVING` is handled by filtering partition isets. `MINUS` and `INTERSECT` are not currently supported in general, though we expect to in the future with extra drop targets on iset widgets. Since isets are sets, our model doesn't support duplicates or ordering. `WHERE` clauses are harder to characterize. `OR`s are only supported over a set of values for a single attribute, as is normal for DQ. `NOT` is not supported explicitly, although set difference can sometimes substitute. Subqueries can't be expressed unless they can be rewritten as joins.

These limitations compared to SQL have rarely impacted our exploration, and are minor compared to other interactive visualization systems.

5. Related Work in Exploratory Data Analysis and Visualization

5.1. Theoretical Analysis

In their theory of "mapping data to visual form," Card, Mackinlay, and Shneiderman [1999] list a set of data transformations that are almost all cases of derived attributes. Transformations such as mean are implemented in Visage by typing formulas. Promotion, defining a binary attribute that holds of entities with a given attribute value, may also be created in Visage by selecting the value with a DQ widget and dragging the result iset to a new attribute widget. Classing, defining an ordinal attribute as ranges of a quantitative attribute, is part of our partition operation. Each partition element has a `category` attribute whose values are the cross product of all the attributes creating the partition. For quantitative attributes, users have control over the ranges that define the ordinal values. Sorting, making vectors, and other structural transformations don't fit our entity-relationship data model. In Visage, sorting is done inside visualizations. Demotion is not needed, because Visage automatically treats ordinal attributes as nominal in visualizations designed for nominal data.

In an information architecture meant to facilitate usable interfaces, Chi [2000] places filtering as the top most operation. This decision is reflected in the prefuse visualization toolkit [Heer et al. 2004]. More generally, faster operations should be architecturally above slower ones. Due to chaining, this is not always the case in VQE. Since filtering or brushing can change the membership of an iset, rebuilding indexes and updating visualizations can significantly slow these operations.

Furnas and Rauch [1998] list multiple ways that Query and Navigation can be integrated, as realized in NaviQue. 1) Applied to VQE, *highlight-directed navigation* would suggest that if one mail folder had lots of hits (i.e. a long dark histogram bar), it might be worthwhile to look at the whole folder. 2) *Navigational specification of queries* and their universes is the focus of this paper, using iset linking and the dual role of isets as a query universe or as a query result. In NaviQue, e-sets serve this purpose. Queries are applied by dropping one e-set (the query) on another (the universe), somewhat like VIQING except that the semantics is similarity-based retrieval rather than join.

Tang, Stolte, and Bosch [2003] argue for a relational data model with built-in ordering among the rows. Their base level system, Rivet [Bosch et al. 2000], visualizes chains of data transformations, allowing the visualization of intermediate results. These intermediate results are nodes in a single data flow, however, rather than independent isets. Indeed, Tang et. al. found that they have not been useful. Rivet also addresses the problem of loading data from multiple data sources,

although it doesn't seem to be able to update them. They believe that a denormalized table is more efficient for analytical processing because it avoids the need for joins. However, their universal join approach will lose track of entities in the case of lossy joins, and doesn't allow for self-joins, for instance. Counts of join records are not necessarily meaningful (see Section 2.3). It is also hard to see how information from multiple data sources could be combined without some sort of join. They advocate splitting the responsibility for transforming data between the database and the visualization system, much as Visage does. The specific transformations they suggest are: group by, sort, filter, aggregate (computing statistics on groups), and merge (putting groups in one table). Having sort as an explicit data transformation is superior to doing it procedurally in visualizations as Visage does. The other operations are included in the ten operations presented above.

In summary, most interactive visualization systems support filtering, projection, and brushing,; a few support partition and joins; and no others support navigation, set operations, or copy and clone. This is likely because they don't keep track of multiple subsets of data.

5.2. System Overviews

In addition to standalone theory, it is useful to look at how current systems support data exploration. Among the many related OLAP, data mining, and visualization systems there is a tradeoff between query expressiveness, dataset size, visualization flexibility, and interactivity. VQE is intended to occupy a useful point on this tradeoff, providing high interactivity for expressive queries over large datasets at a cost of limited visualization flexibility. In general, VQE is distinguished by an emphasis on creating multiple isets rather than operating on entire datasets; visualization of summary statistics rather than just being record oriented; and, in combination with SAGE, treating visualization as orthogonal to data manipulation. The query semantics issues discussed above are most relevant for visual query languages, which have the potential to be more expressive than menus and easier to use than text [Ahlberg et al. 1992; Catarci and Santucci 1995].

Polaris [Stolte et al. 2002], DEVise [Livny et al. 1997], and DataSplash [Olston et al. 1998] are most similar to VQE in approach to integrating interactive query and visualization. They have less expressive query languages, and haven't been scaled to massive datasets. DataSplash is notable for the ease of creating nested queries using drag and drop (it doesn't allow dragging individual entities however). We strive to make the VQE visual query language as close to this ideal as possible, while recognizing that an explicit declarative structure independent of visual presentation has advantages for supporting analysts over time.

Dynamic Query [Ahlberg et al. 1992] initiated a paradigm shift from batch database queries to interactive filtering of visualizations for single tables of data. Ioannidis [1996] extended DQ to queries with joins. Our notion of "iset widgets"

further extends the interactive paradigm to drag and drop interaction with join queries. VQE supports more efficient DQ for these queries [Derthick et al. 1999] than the grid file approach advocated by Ioannidis. The Aggregate Manipulator [Goldstein et al. 1994] previously used partition, DQ, and visualization for data exploration. Aggregates are similar to isets, although once defined, aggregate membership did not change. VQE supports dynamic filtering of iset membership and uses a more flexible visualization-based interface for iset definition. DEVise [Livny et al. 1997] offers variable levels of partition, and it has been integrated with the BIRCH clustering system to extend SQL aggregation operations. DEVise filtering is done on graphical data, rather than the underlying domain data as in VQE. Therefore, only the attributes that are expressed in the visualization can be used. Graphical level linking between visualizations that show different attributes must be accomplished with master/slave links, which limit the slave records to those with a matching record in the master. In other systems linking by brushing is built in to the architecture rather than specified for particular pairs of visualizations.

VQE's evolutionary process of forming isets is similar to that of IMACS [Brachman et al. 1993], which allows marketing analysts to segment customers and track their behavior over time. IMACS supports joins (called role chains in their frame terminology) so that customers can be segmented by properties of their purchases. Binning of quantitative attributes can be done graphically, but all other query parameters are specified in a textual query language, QL. VQE consciously follows the IMACS approach, improving on it with respect to dataset size and interactivity. Similar to IMACS, as the data changes over time, those entities that satisfy the definition of an iset change.

VQE's node and link visual query syntax is a simplified version of GQL [Papantonakis and King 1995], which has the expressive power of SQL. GQL only returns a non-interactive table of results, however it uses links rather than containment to represent attributes. The node and link syntax is similar to that of Microsoft Access, which like GQL presents the query result separately from the query rather than making the query itself a visualization-based interface.

DataSplash's VIQING [Olston et al. 1998] is noteworthy for supporting joins purely through lightweight direct manipulation, without requiring an explicit visual language such as VQE's node-and-link query graphs. To join two isets, one is dropped on the other. The relationship joined on is implicit, so there is no way to specify with direct manipulation, for instance, that you intend to link email messages to their recipients rather than their senders. SQL can be used to manually specify the join predicate in this case. VIQING join queries are always visualized using portals. Each portal is a copy of the dropped visualization, and there is one portal for every record of the dropped-on visualization. This works well for small datasets with one-to-many relationships, but not so well with many-to-many relationships. VQE is optimized for large datasets where this record-oriented join visualization is not appropriate. As future work, Olston *et. al.* plan to handle

large datasets, saying that the DataSplash design is orthogonal to this issue. However this will require the ability to aggregate and/or select records before visualizing all the records, which may negatively impact its current thoroughly direct manipulation paradigm and make it more like VQE/SAGE. An earlier version of DataSplash used a node-and-link visual query syntax, but it was discarded because users found it too hard to use. VQE attempts to avoid this problem by making the nodes represent isets rather than abstract schema objects. As noted above, VQE's explicit query language allows increased control and serves as a record of the operations performed, which can aid working over time.

North and Shneiderman [2001] have defined a taxonomy of multiple window coordination, in which operations performed in one window affect the other window. Their *selecting items to items* accounts for brushing. They point out that this can be generalized to selecting items in the second window that are *related* to those in the first. In VQE, the query graph specifies the relationship among entities. Then, nested SAGE pictures visualizing entities of each type are automatically linked by brushing tuple entities.

DOODLE [Cruz 1992] is more oriented to visualization design than querying, yet does support joins and aggregation. Querying is not incremental, and only a single result set is returned, which is limited to tens of thousands of records

5.3. Comparison of expressive power and ease of use on example query

Few other visual query interfaces can answer the question "Who has md phoned but never emailed?". IMACS lacks a `NOT` operator, because it would introduce non-monotonicity into the classification reasoning. DOODLE is based on F-Logic [Balaban 1995], which was developed to model description languages like IMACS, and thus also lacks a `NOT` operator. (Of course, VQE's visual language cannot express `NOT` either, but drag and drop can be used to generate SQL containing `NOT`.)

VIQING would generate a tree-structured nesting for each join that leads to 66 Contacts frames at the second level, which collectively have 2002 leaf messages. The join operation is tightly integrated with the nested portal visualization, which is intended to show the join records grouped by phone call. There is no way to collect all the contacts that can be joined to at least one email message. Microsoft Access' visual interface is not much help, because the nested query must be typed in SQL syntax as part of the "criteria" for the `contacts.id` attribute. By defining the nested query as a separate named view it is possible to take better advantage of the visual interface. Either way, it is certainly not the case that incremental feedback is given to indicate whether the user is on the right track.

GQL has the full expressiveness of SQL. It intuitively maps nested queries to containment in boxes. Figure 21 shows the GQL representation of the example query, which appears to be simpler than any of the other representations to understand or construct. Some of the simplicity is deceptive, because there are 10 additional kinds of boxes like the NOT

EXISTS subquery box in the figure, each denoted by different background patterns and the presence or absence of tick marks. Thus there are multiple semantically distinct queries that would look very much like the one in the figure. One can imagine an interactive version of GQL that gives feedback about result sets for each node or box as the query is incrementally constructed.

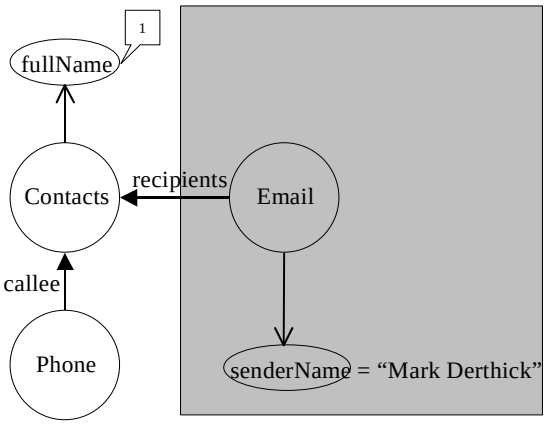

Figure 16 GQL representation of "Who has md phoned but never emailed?" The shaded box represents a `NOT EXISTS` subquery. The callout box labeled ‘1’ indicates that `fullName` is the first (and only) column returned by the query.

6. Future Work: Declarative Representation of Drag-and-Drop Set Operations

Section 3.5 illustrated the power of drag and drop to remove the iset of contacts md phoned but never emailed. However this operation was not reflected in the query graph. Further, the only supported set operations are set union of an arbitrary iset with a universe iset, and set difference between a universe iset and its result iset. We plan to add a third summary iset to the summary line of iset widgets (red boxes in Figure 17). This represents the complement of the universe with respect to all instances of the entity type. Set intersection is accomplished with drag and drop set union of the complements. Set difference is accomplished by dragging any iset to a complement iset.

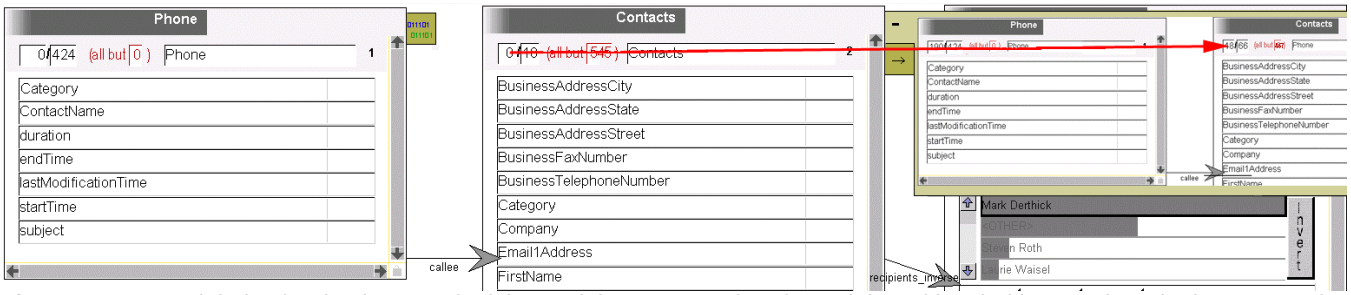

Figure 17 Proposed design for showing a stack of drag-and-drop set operations for each iset widget (gold squares). Selecting an operation shows the iset dragged to/from in context, with an arrow connecting the drag origin and destination.

A history of set operations will be visualized as a column of "attachments" to the iset widget. The lowest attachment represents the way the original universe iset was defined, and each attachment higher in the stack represents a set operation on the universe. For instance, in Figure 22 the `Phone` iset was defined by cloning ( , meant to suggest copying the underlying iset) another iset comprising all calls. The `Contacts` iset was defined originally by navigation (→), and then was modified by set difference (-). If the attachment is expanded, it shows the VQE representation of the universe iset and a blue arrow connecting that iset to the one it was dropped on (for set union), or a red arrow to it from the iset it was dragged from (for set difference). Figure 16 illustrates this for the expansion of the final operation of the phone-no-email example.

A procedural text representation may be useful as well, either for writing macros directly or as a more compact representation of the history than Figure 16. The main problem is finding descriptive referring expressions for objects on the screen. A possibility is relying on iset widget labels. A script similar to Figure 18 could generate Figure 13 and label the iset widgets `AllPhoneCalls`, `ContactsCalled`, and `EmailToContactsCalled`. Conversely, if a user labeled the iset widgets this way, Figure 23 could be generated automatically.

```
CopyCloneDrag from Desktop.AllPhoneCalls to
      VQE1 as new IsetWidget: AllPhoneCalls
Navigate from VQE1.AllPhoneCalls via callee to
      VQE1 as new IsetWidget: ContactsCalled
Navigate from VQE1.ContactsCalled via recipients_inverse to
      VQE1 as new IsetWidget: EmailToContactsCalled
Filter on VQE1.EmailToContactsCalled.SenderName: SenderFilter
Invert SenderFilter
Toggle SenderFilter {'Mark Derthick'}
Drag from VQE1.ContactsCalled.result to Desktop
CopyCloneDrag from VQE1.ContactsCalled to Desktop.ResultIsHere!
```

Figure 18 Possible style of scripting to generate the phone-no-email example.

7. Conclusion

7.1. Evaluation: Expressive Power vs. Ease of Use

The ten operations, iset pairs as persistent intermediate results, and derived attributes provide more expressive power than other interactive visualization systems. There is only indirect evidence that 1) this level of expressivity is important for a large segment of analysts, or that 2) Visage and VQE effectively support this power. In regard to the second question, one informal study found that developers could construct information appliances interactively [Derthick and Roth 2001b]; another unpublished informal study found that some US Army analysts preferred Visage to their current tools for identifying transportation bottlenecks. Developers at Maya and CMU have used Visage and VQE on dozens of datasets, from public health surveys, to geographical transportation planning, digital video libraries, and evolutionary trees. This flexibility gives hope that they will be broadly applicable, but much more evaluation remains to be done.

In regard to the first question, most data mining, statistics, and visualization software focuses on analyzing data in a single table. Any joining or aggregation happens at import time only. There is evidence that more expressive power is required for some users. The Sloan Digital Sky Survey is recording about 500 attributes for 200M stars and galaxies. It includes 95 tables and views. A web site intended for astrophysicists who do not necessarily have computer science training supports interactive queries over this data by typing SQL [SDSS.org 2004]. Most of the help provided consists of example queries that they can copy and edit; joins and aggregation are heavily represented. Of 100,000 queries posted from 7 May to 22 May 2004, 7000 involved joins (200 of them outer joins), 23 involved `GROUP BY`, 1500 used inequalities, 33 used `NOT`, 1000 used `OR` (in all but 48 of these the disjuncts were predicates on a single attribute that could be expressed using DQ), and none used `UNION`, `MINUS`, or `INTERSECT`.

In work meant to capture the cognitive process behind choosing airplane flights, Casner [1991] used distinct variables to represent entities subject to what amounts to join constraints. The arrival airport and time must relate appropriately to the departure airport and time of the other. The importance of navigation in user interfaces also suggests that a declarative model of users' interaction will involve an operation like join.

Older information retrieval systems intended for expert users, such as Dialog and Lexis-Nexis, treated intermediate results as set expressions that could be reused in subsequent searches. This becomes problematic in a true object-oriented system, because modifying a query loses track of the original. For this reason DLITE [Cousins et al. 1997] doesn't support set union or difference. The ability presented here to clone rather than just copy addresses this difficulty.

All of Jakob Nielson's heuristics for "Extending Task Analysis to Predict Things People Many Want to Do" by "Goal Composition" [1994] benefit from having first class isets that can be copied and cloned, that are interactively updated, and whose visual context in iset widgets reminds users of their meaning:

- Generalization Mechanisms
  - Multiplexing: *iset manipulation is all about doing the same thing to multiple entites.*
  - Reuse: *iset widgets can be emptied and new data added.*
  - Supergoaling: *results of multiple queries can be combined in support of a higher level goal.*
- Integration Mechanisms
  - Interleaving: *intermediate results remain available as users work on other analyses.*
  - Suspension and Postponement: *the complete interface state can be saved and reloaded.*
  - Result Passing: *linking results to new queries is a central feature.*
  - Automated Use: *scripting (Section 6) would allow other programs to invoke these operations.*
- User Control Mechanisms
  - Monitoring: *continuous feedback supports user monitoring.*
  - Result Investigation: *it is easy to explore and verify results.*
  - Recording and Retrieving: *query graphs and history visualization [Derthick and Roth 2001a] do this.*
  - Alternative Enumeration: *menus help support this.*
  - Reverting: *Visage supports tree-structured infinite undo [Derthick and Roth 2001a].*
  - Modification and Editing: *Visage supports selective redo [Derthick and Roth 2001a]and macros [Derthick and Roth 2001b].*

7.2. Shortcomings

VQE imposes a great deal of structure on the exploration process. Data manipulation supported by the ten operations is much simpler than writing SQL expressions. However, non-supported manipulations require users to manually assign SQL to isets, and there is no smooth learning path. For expressing navigation, we claim that SQL does not fit the user's domain model, and is best hidden. Designers should always think carefully before giving the user a false model of the system's information architecture. Indeed, Visage exhibits anomalous behavior in the case of too much brushing due to tuple entities. Imprecise iset counts for multi-entity queries are a source of user confusion since approximate algorithms must be used for efficiency. Building indexes is slow (ten seconds for 100,000 records and 5 sliders), which disrupts the flow of the analysis when the user adds new sliders or creates new large isets.

A more general problem with VQE is the lack of affordances. Even for a Visage user who knows the gestures for the basic operations like navigation, copying, and cloning, VQE introduces many semantic extensions. Some operations do carry over: VQE does support the normal Visage navigation gesture from iset widgets, even though it offers the menu of arrows as a preferred alternative. Doing a join rather than a navigate can only be done with the arrows. Most of the VQE extensions support special cases of drag and drop. Set difference/union/intersection by dragging in/out of iset widgets does

not hold for isets outside VQE. Attribute definition, which is made much easier by drag and drop, is only possible in VQE. Although it is natural to drag a company attribute to an email iset widget to define a `sender_company` attribute for messages, there is no indication that this is possible. Dragging an iset widget to denote the hidden ID attribute is even less obvious. Thus, VQE remains a tool for experts.

Interactive feedback may help. For instance, when an object is dragged, all the objects it can be dropped on could highlight in a color that codes whether it will be moved or copied, and whether it will be added as a separate object, transformed to a new distinct object, or affect the current object in some way. Hovering over a highlighted object could give more detail in a pop-up window. Another way to break down the barrier to non-expert use may be creating VQE queries from an example, followed by "generalization by deletion" [Derthick and Roth 2001b]. In any case, appliances and briefings generated by an expert can be reused by anyone.

7.3. Contributions

The primary contribution of this paper is using direct manipulation of meaningful entities to support sequences of interactive exploratory operations for large datasets including multiple entity types. This includes making it easy to perform set operations on isets without using syntax, navigate to form new context-sensitive isets, link successive intermediate results allowing bi-directional propagation, and define new attributes. Queries and visualizations are linked, allowing the user to think either in terms of an iset's intention or extension and to change perspectives on the fly. These principles support the following features:

- Queries are expressed using 10 direct manipulation operations, using a visual language that makes relationships explicit.
- Query chains linked by sharing helps visualize the analysis process.
- Queries (isets) represent concrete entities, as reflected in continuously updated histograms.
- Queries can be created by drag and drop operations from visualizations, and visualizations can be created from queries.
- All visualizations created during an exploration session remain coordinated as the query is modified.
- Entities of one type can be filtered based on attributes of entities of another type.
- Visualizations can combine attributes from multiple entity types.
- An exploration session consisting of queries and visualizations can be saved independently of any data, and reused on a different data set.

Acknowledgements

This material is based on work supported by the National Science Foundation under Cooperative Agreement No. IRI-9817496 and by DARPA contract DAA-1593K0005. Jake Kolojejchick and Steve Roth contributed significantly to the early design of VQE.

References

Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R. and Pinkerton, M. 1997. Cyberguide: a mobile context-aware tour guide. *Wireless Networks* 3, 5, 421 - 433.

Ahlberg, C., Williamson, C. and Shneiderman, B. 1992. Dynamic Queries for Information Exploration: An Implementation and Evaluation. In *Human Factors in Computing Systems (CHI)*. Monterey, CA. ACM Press, 619-626.

Balaban, M. 1995. The F-logic Approach for Description Languages. *Annals of Mathematics and Artificial Intelligence* 15, 1, 19-60. http://citeseer.nj.nec.com/balaban93flogic.html

Becker, R. A. and Cleveland, W. S. 1987. Brushing Scatterplots. *Technometrics* 29, 2, 127-142.

Bell, G. and Gray, J. 2000. Digital Immortality. Microsoft Research: San Francisco, CA. http://research.microsoft.com/~gray/papers/MSR_TR_00_101_Digital_Immortality.pdf

Bentley, J. L. 1979. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering* 5, 4, 333-340.

Bosch, R., Stolte, C., Tang, D., Gerth, J., Rosenblum, M. and Hanrahan, P. 2000. Rivet: A Flexible Environment for Computer Systems Visualization. *Computer Graphics* 34, 1, 68 - 73. http://graphics.stanford.edu/papers/rivet_cg/rivet.pdf

Brachman, R. J. and Schmolze, J. G. 1986. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9, 2, 171-216.

Brachman, R. J., Selfridge, P. G., Terveen, L. G., Altman, B., Borgida, A., Halper, F., Kirk, T., Lazar, A., McGuinness, D. L. and Resnick, L. A. 1993. Integrated support for data archaeology. *International Journal of Intelligent and Cooperative Information Systems* 2, 2, 159-185.

Card, S. K., Mackinlay, J. D. and Shneiderman, B. 1999. Introduction. In *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann.

Casner, S. M. 1991. A Task-Analytic Approach to the Automated Design of Graphic Presentations. *ACM Transactions on Graphics* 10, 2, 111-151.

Catarci, T. and Santucci, G. 1995. Diagrammatic vs Textual Query Languages: A Comparative Experiment. In *Proceedings of the IFIP W.G. 2.6 Working Conference on Visual Databases*. Lausanne, 57-74.

Chi, E. H. 2000. A Taxonomy of Visualization Techniques Using the Data State Reference Model. In *InfoVis '00*, 69-75.

Cousins, S. B., Paepcke, A., Winograd, T., Bier, E. A. and Pier, K. A. 1997. The Digital Library Integrated Task Environment (DLITE). In *Proceedings of the 2nd ACM International Conference on Digital Libraries*. Philadelphia, PA, USA. ACM, 142-151.

Cruz, I. F. 1992. DOODLE: A Visual Language for Object-Oriented Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*. San Diego, California, 71-80.

Derthick, M., Harrison, J., Moore, A. and Roth, S. F. 1999. Efficient Multi-object Dynamic Query Histograms. In *Proceedings of IEEE Information Visualization Symposium (InfoVis'99)*. IEEE Press, 84-91. http://www.cs.cmu.edu/~sage/PDF/IV99.pdf

Derthick, M., Kolojejchick, J. A. and Roth, S. 1997. An Interactive Visual Query Environment for Exploring Data. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*. Banff, Canada. ACM Press, 189-198. http://www-2.cs.cmu.edu/~sage/PDF/Query.pdf

Derthick, M. and Roth, S. F. 2001a. Enhancing Data Exploration with a Branching History of User Operations. *Knowledge Based Systems* 14, 1-2, 65-74. http://www.cs.cmu.edu/~sage/Papers/KBS/KBS.pdf

Derthick, M. and Roth, S. F. 2001b. Example-based generation of custom data analysis appliances. In *Proceedings of Intelligent User Interfaces (IUI '01)*. Santa Fe, NM, 60-67. http://www.cs.cmu.edu/~sage/Papers/IUI00/IUI00.pdf

Fayyad, U. M., Piatetsky-Shapiro, G. and Smyth, P. 1996. Data Mining and Knowledge Discovery in Databases: An overview. *Communications of the ACM* 39, 11, 27-34.

Flajolet, P. and Martin, G. N. 1983. Probabilistic Counting. In *Foundations of Computer Science*. IEEE Press, 76-82.

Freeman, E. and Gelernter, D. 1996. Lifestreams: A Storage Model for Personal Data. *ACM SIGMOD Record* 25, 1, 80-86.

Furnas, G. W. and Rauch, S. J. 1998. Considerations for Information Environments and the NaviQue Workspace. In *ACM Digital Libraries*, 79-88. http://www.si.umich.edu/~furnas/POSTSCRIPTS/navique4.ps

Goldstein, J., Roth, S. F., Kolojejchick, J. and Mattis, J. 1994. A Framework for Knowledge-Based, Interactive Data Exploration. *Journal of Visual Languages and Computing* 5, 4, 339-363. http://www.cs.cmu.edu/~sage/PDF/Framework.pdf

Hauptmann, A. and Lin, W.-h. 2001. Video and Audio Analysis for Remembering Conversations. In *Automatic Speech Recognition and Understanding Workshop*. Madonna di Campiglio, Italy, 9-13.

Heer, J., Card, S. K. and Landay, J. A. 2004. prefuse: a toolkit for interactive information visualization. http://jheer.org/publications/2004-prefuse.pdf

Houghton, R. 1999. Named Faces: Putting Names to Faces. *IEEE Intelligent Systems Magazine*, 14, 5, 45-50.

Ioannidis, Y. 1996. Dynamic information visualization. *SIGMOD record* 25, 4, 16-20.

Livny, M., Ramakrishnan, R., Beyer, K. S., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J. and Wenger, R. K. 1997. DEVise: Integrated Querying and Visual Exploration of Large Datasets. In *Proceedings ACM SIGMOD International Conference on Management of Data*. Tucson, Arizona. ACM Press, 301-312. http://citeseer.nj.nec.com/livny97devise.html

Mackinlay, J. D. 1986. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics* 5, 2, 110-141. http://www2.parc.com/istl/projects/uir/pubs/items/UIR-1986-02-Mackinlay-TOG-Automating.pdf

Neuwirth, C. M., Morris, J. H., Regli, S. H., Chandhok, R. and Wenger, G. C. 1998. Envisioning communication: task-tailorable representations of communication in asynchronous work. In *Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work*. ACM Press, 456-463.

Nielsen, J. 1994. Goal Composition: Extending Task Analysis to Predict Things People *May* Want to Do. http://www.useit.com/papers/goalcomposition.html

North, C. 2001. Multiple Views and Tight Coupling in Visualization: A Language, Taxonomy, and System. In *Proc. CSREA CISST 2001 Workshop of Fundamental Issues in Visualization*, 626-632.

Olston, C., Stonebraker, M., Aiken, A. and Hellerstein, J. M. 1998. VIQING: Visual Interactive QueryING. In *Proceedings of the 14th IEEE Symposium on Visual Languages*. Halifax, Canada, 162--169.

Papantonakis, A. and King, P. J. H. 1995. Syntax and semantics of GQL, a graphical query language. *Journal of Visual Languages and Computing* 6, 1, 3-25.

Provost, F. and Fawcett, T. 2001. Robust Classification for Imprecise Environments. *Machine Learning* 42, 3, 203-231. http://pages.stern.nyu.edu/~fprovost/Papers/rocch-mlj.pdf

Rayson, J. 1999. Aggregate Towers: Scale Sensitive Visualization and Decluttering of Geospatial Data. In *IEEE Symposium on Information Visualization (Infovis)*. San Francisco, California. IEEE, 92-99.

Rhodes, B. J. and Starner, T. 1996. Remembrance agent: A continuously running automated information retreival system. In *Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology (PAAM)*. London. Practical Application Co., Ltd, 487-495.

Roth, S. F., Chuah, M. C., Kerpedjiev, S., Kolojejchick, J. A. and Lucas, P. 1997. Towards an Information Visualization Workspace: Combining Multiple Means of Expression. *Human-Computer Interaction Journal* 12, 1-2, 131-185. http://www.cs.cmu.edu/~sage/PDF/Towards.pdf

Roth, S. F. and Mattis, J. 1990. Data characterization for intelligent graphics presentation. In *Proceedings of the Conference on Human Factors in Computer Systems (CHI'90)*. ACM Press, 193-200.

SAS Institute. 2001. JMP: The Industrial Strength Discovery Tool. http://www.jmpdiscovery.com/index.html

SDSS.org. 2004. Searching for Data: A Tutorial. http://cas.sdss.org/dr2/en/help/howto/search/

Shneiderman, B. 1982. The future of interactive systems and the emergence of direct manipulation. *Behavior and Information Technology* 1, 3, 237-256.

Silicon Graphics Inc. 2004. MineSet. http://www.sgi.com/products/appsdirectory.dir/linux/products/m/958629.html

Smeaton, A. F., Over, P., Costello, C., Vries, A. P. d., Doermann, D. S., Hauptmann, A. G., Rorvig, M. E., Smith, J. R. and Wu, L. 2002. The TREC2001 Video Track: Information Retrieval on Digital Video Information. In *Sixth European Conference on Digital Libraries (ECDL 2002)*. Rome, Italy, 266-275. http://www.informedia.cs.cmu.edu/documents/ecdl02_trec01.pdf

Stolte, C. and Hanrahan, P. 2000. Polaris: A System for Query, Analysis and Visualization of Multi-dimensional Relational Databases. In *Proceedings of IEEE Information Visualization Symposium (InfoVis'2000)*. IEEE Press, 5-14. http://graphics.stanford.edu/papers/polaris/

Stolte, C., Tang, D. and Hanrahan, P. 2002. Polaris: A System for Query, Analysis and Visualization of Multi-dimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1, 52-65.

Tang, D., Stolte, C. and Bosch, R. 2003. Design Choices when Architecting Visualizations. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*. Seattle, Washington. IEEE, 41- 48. http://ieeexplore.ieee.org/iel5/8837/27965/01249007.pdf?isNumber=27965&arnumber=1249007&prod=CNF&arSt=+41&ared=+48&arAuthor=Tang%2C+D.%3B+Stolte%2C+C.%3B+Bosche%2C+R.

Tufte, E. R. 1983. *The Visual Display of Quantitative Information*: Graphics Press. 197 pages.

Tweedie, L., Spence, R., Dawkes, H. and Su, H. 1996. Externalising Abstract Mathematical Models. In *Human factors in computing systems (CHI)*. Vancouver, Canada. ACM, 406-412.

Velleman, P. F. 1993. *Learning Data Analysis With Data Desk*. Revised Edition ed: W.H. Freeman & Company. 64 pages.

Wactlar, H. D., Christel, M. G., Hauptmann, A. G. and Gong, Y. 1999. Informedia Experience-on-Demand: capturing, integrating and communicating experiences across people, time and space. *ACM Computing Surveys* 31, 2es.

Whittaker, S. and Sidner, C. 1996. Email overload: Exploring personal information management of email. In *Proceedings of CHI '96*. ACM, 276-283.
Wilkinson, L. 1999. *The Grammar of Graphics*: Springer. 408 pages.