Coordinating Declarative Queries with a Direct Manipulation Data Exploration Environment

Mark Derthick, Steven F. Roth, John Kolojejchick
Carnegie Mellon University Robotics Institute
{mad+, jake+, roth+}@cs.cmu.edu

Abstract

Interactive visualization techniques allow data exploration to be a continuous process, rather than a discrete sequence of queries and results as in traditional database systems. However limitations in expressive power of current visualization systems force users to go outside the system and form a new dataset in order to perform certain operations, such as those involving the relationship among multiple objects. Further, there is no support for integrating data from the new dataset into previous visualizations, so users must recreate them. Visage's information centric paradigm provides an architectural hook for linking data across multiple queries, removing this overhead. This paper describes the addition to Visage of a visual query language, called VQE, which allows users to express more complicated queries than in previous interactive visualization systems. Visualizations can be created from queries and vice versa. When either is updated, the other changes to maintain consistency.

1. Introduction

Exploratory data analysis is an iterative process where high level questions lead to specific queries whose answers are examined for interesting patterns. These in turn suggest new questions. To facilitate this kind of exploration, we would like to provide the analyst rapid, incremental, and reversible operations giving continuous visual feedback. However we also need the expressive power to reorganize the data on the fly, to juxtapose objects according to diverse criteria, and visualizations to show relationships among properties of these different objects. In short, we want both the ease of use of direct manipulation systems and the power of database query systems. This need is recognized, yet in current systems the architecture for connecting them is a feedforward batch stream from query to visualization system, each having a separate interface.

2. Current Systems

2.1. Database Management Systems

Current widely used database user interfaces fall primarily into two classes: SQL for programmers, and forms-based interfaces for others. The former is hard to use, and the latter is not very powerful. Using forms, an application builder must anticipate all useful classes of queries, parameterize them, and design a form to provide the parameters. For exploratory data analysis, queries cannot be anticipated, yet we would like non-programmers to express requests for data. Although not yet widely deployed, database research is advancing ease of use in several directions. First, visual query languages aim to be simpler to use than textual languages like SQL. Some are based on the Entity-Relationship or other object-oriented models, which are closer to the user's conception of the problem domain than the implicit relationships in SQL's relational model. Second, database management systems (DBMS) are also being integrated with visualization systems and report generators for output that is easier to assimilate. However, the mode of user interaction is still batch. Based on the visualized output, a user formulates a new query to get a new output, rather than manipulating the visualization itself to retrieve different data.

2.2. Interactive Visualization Systems

Visualization systems have the opposite problem: they are easy to use but their interfaces are insufficient for expressing queries. They can not be used to pose queries reflecting multiple objects, aggregation, quantification, disjunction, negation, or arithmetic and logical functions of attributes. This is largely because it is viewed as the job of the DBMS to create a table, and the visualization system's job to show all and only the data in the table. In currently deployed systems, visualizations do not provide query functions themselves. Although some drill-down capability may be available, changing a query requires a distinct interface.

Further, each time a query is changed and processed, a new table is sent to the visualization system, which can not coordinate visualizations from the different tables even if the underlying data is identical. For instance, if interface objects in one visualization are painted or filtered, the corresponding objects in visualizations derived from other tables will be unaffected.

0-8186-8189-6/97 $10.00 © 1997 IEEE

Recent research has increased the amount of querying that is possible within a visualization to support exploration. Direct manipulation systems like IVEE [1] and Visage [11] provide rapid incremental construction of "queries" using Dynamic Query sliders, Alpha Sliders, painting, and visualization selection. Ahlberg *et al* [2] lists the following advantages of direct manipulation:

1. Continuous visual representation of objects and actions of interest.
2. Physical actions instead of complex syntax.
3. Rapid, incremental, reversible operations whose impact on the object of interest is immediately visible.
4. Layered or spiral approach to learning that permits usage with minimal knowledge.

Their experiments showed that students answered questions faster and more accurately with a direct manipulation interface than a text-only one, or one with graphic display and fill-in-the-blank queries.

But what you see is all you can manipulate. Querying about aggregate properties of sets of objects is difficult, as is expressing queries based on relationships among multiple objects or properties of groups of objects. Direct manipulation interfaces are generally hard to use to refer to data that is not currently visible.

2.3. Visage

Visage is an *information centric* [9, 11] user interface environment for data exploration and for creating interfaces to data-intensive applications. Data objects are represented as first class interface objects that can be manipulated using a common set of basic operations, such as drill-down and roll-up, drag-and-drop moves, copy, and dynamic scaling. These operations are universally applicable across the environment, whether graphical objects appear in a hierarchical table, a map, a slide show, in a query or other application user interface. Furthermore graphical objects can be dragged across application UI boundaries. In addition to capabilities of previous interactive visualization systems, Visage also includes the SAGE system [10, 11] for automated design of visualizations that integrate many attributes. SAGE is a visualization server to Visage, which renders SAGE graphic designs so that they are subject to all direct manipulation operations. Integrating the visualization system directly with an underlying database, rather than just deriving visualizations from otherwise isolated tables, is key to coordinating visualization applications with the other components of an exploratory data analysis environment.

Visage includes several ubiquitous exploration operations that are related to querying. A user can *navigate* from the visual representation of any database object to other related objects. For instance, from a graphical object representing a real estate sales agent, one can navigate to all the houses listed by that agent. It is also possible to *aggregate* database objects into a new object, which will have properties derived from its elements. For instance, we could aggregate the houses listed by John and look up their average size or "recompose" this set into sub-aggregates based on neighborhood or number of rooms (i.e. all Shadyside houses listed by John with 8 rooms).

However these operations are only specified procedurally, that is, by a sequence of direct manipulation operations. There is no explicit query. Once a user has navigated from John to his houses and aggregated them, he must repeat the operations to do the same for Jill's houses, or to repeat for John's houses next month when the data may have changed. Just as sliders visually indicate the current filtration range for parameterized queries, we would like a declarative and manipulable visual query representation that also applies to these *structural* operations. We would like to dynamically and incrementally change or reuse some or all portions of a sequence of data exploration operations.

2.4. VQE: Visual Query Environment

VQE is a visual query environment within Visage for representing operations explicitly. It enables an analyst to construct complex queries during data exploration and reuse them later. VQE queries can express relationships among object sets, support navigation among objects, and denote aggregation. They combine this level of expressiveness with dynamic query interfaces for range selection. Furthermore, queries are fully integrated with the rest of Visage: not only can query results be dragged to other visualizations, but objects from visualizations can be dragged into queries (i.e. be the input to query expressions). User directed changes to queries and visualizations are immediately reflected in each other.

3. Example

3.1. Schema Browser

To illustrate the system, we use a fictitious database of house sales used by a group of Pittsburgh real-estate agents describing clients and sales information for three neighborhoods in 1989. Figure 1 (top) shows an entity-relationship (ER) diagram of the database. It serves as an interface for browsing database structure and initiating queries. Data types are shown as rectangular nodes, and attributes of objects are listed inside the node. For example, the attributes of houses include lot size, number of rooms, address, and neighborhood. Binary relationships between objects are shown as links. The appearance of the links shows functional dependencies. Links with one line diverging into multiple ones indicate one-many relationships; those with multiple lines in both directions indicate many-many relationships. A link with a single line would indicate a one-to-one relationship. For example, in this database, only one person can be the buyer_agent in a given sale, although many sales can have the same person

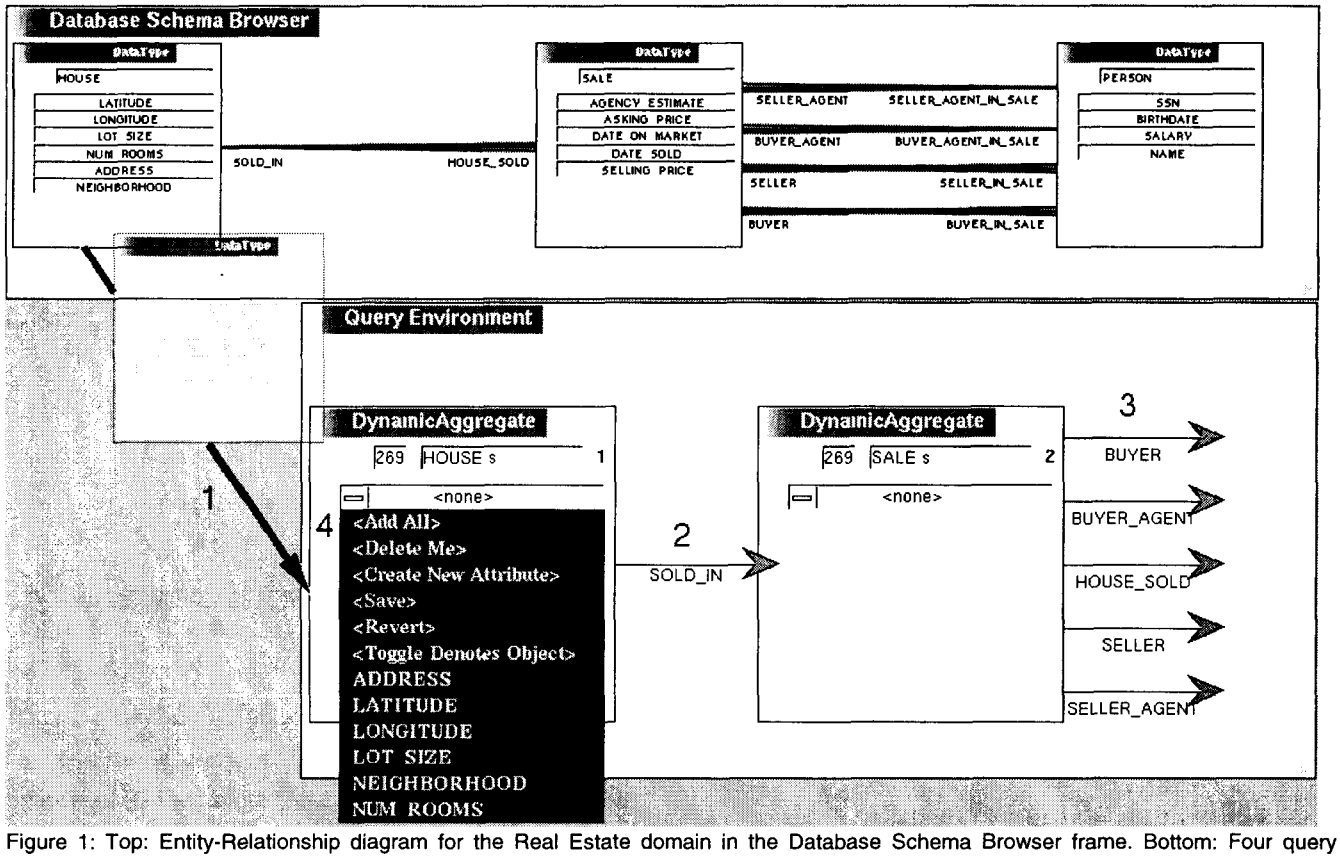

Figure 1: Top: Entity-Relationship diagram for the Real Estate domain in the Database Schema Browser frame. Bottom: Four query construction operations have been performed: 1) A copy of the House data type has been dragged from the ER diagram to QueryEnvironment (VQE), where it becomes a *dynamic aggregate* of Houses; 2) navigation along the "sold_in" relationship to Sales; and 3) a set of relationships emanating from Sales has been displayed as arrows as options to navigate further; 4) a menu of attributes for the House dynamic aggregate is exposed.

as the buyer_agent. The role served by a relationship for an object is labeled next to the data type. For instance, the relationship between sales and houses serves the house_sold role of the sale, while it is the sold_in role of the house.

ER diagrams are useful for browsing because they show all the data types in the database and their attributes. They also show all the relationships and their domain and range. However, it's important to emphasize that their purpose here is just to show the structure of the database and as a starting point for navigating with actual data sets.

The remainder of the example illustrates how a real estate agent might create and modify a query during data exploration. It illustrates a solution to a fundamental limitation of current interactive visualization systems, related to the integration of multiple distinct data objects (i.e. a join in relational database terms). Previous systems like IVEE and Visage enable users to assign sliders to filter a set of data objects based on one or more attributes (e.g. to filter houses based on number_of_rooms or lot_size). Both systems also enable users to visualize relationships among multiple attributes using graphical properties of objects in charts (e.g. the relationship between number_of_rooms and lot_size in a plot chart). However, users often need to view relationships among attributes that are distributed across multiple related objects. For example, one might want to filter houses based on the salary of the buyers of the houses. But *salary* is an attribute of *person,* not *house.* In our database, *houses* are related to *sales,* which are related to *persons.* Therefore, we need a way to specify the "path" from house to person to use attributes of one to filter the other or to view relationships between attributes of different objects (e.g. *person salary* vs. *sale price* in Figure 3). In general, it is not possible to anticipate every combination of data objects a user will want to integrate or to create the "universal" relation that joins all data into one object. Therefore VQE contains a navigation mechanism to dynamically construct paths among object sets to integrate their attributes.

The following example illustrates:

- Navigation among sets of objects of different types.
- Visualization of attributes from multiple object types in a single graphic.
- UI techniques for assigning data attributes to be visualized to graphical properties.
- Extension of dynamic query filter techniques to control multiple objects sets.
- Coordination among visualizations derived from different queries.
- Dynamic definition of new data attributes.

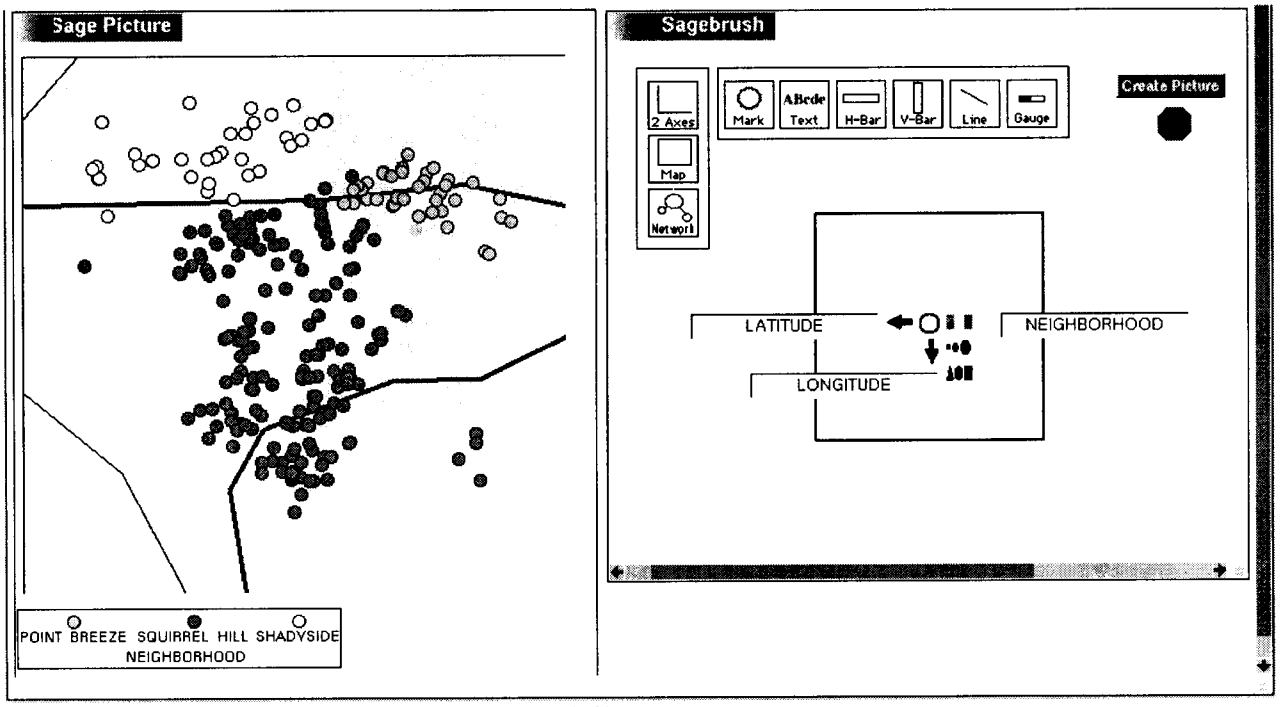

Figure 2: A SageBrush sketch (right) and the resulting picture (left).

3.2. Integrating Multiple Objects

Figure 1 (bottom) shows the query after one navigation step has been completed, and another is beginning. First a copy of the House data type was dragged from the ER diagram into the VQE frame, where it became a dynamic aggregate. The dynamic aggregate represents a set of houses, and serves as a locus for browsing and querying operations on the elements of the set. The data type of the dynamic aggregate is preceded by its cardinality; in this database there are 269 houses. In this case, the data type for houses was dragged into the Query Environment, so all objects of type *house* in the database formed a dynamic aggregate. It is also possible to drag subsets of particular houses from visualizations to form the aggregate. A mouse operation on a dynamic aggregate brings up a choice of relationships for navigation, displayed as arrows (in this case, there is only one relationship for houses: *sold_in*). The user selected the arrow, which produced the second dynamic aggregate, labeled "Sales". The choices for navigation from Sales has also been displayed. Arrows show the direction of navigation, and the labels depend on this direction as described above. For instance, we are navigating away from Sales, so the top relationship is labeled *buyer* rather than buyer-in-sale. The user will select the "buyer" arrow in order to build a third dynamic aggregate of Persons who were buyers in the set of sales (Figure 3 shows all three dynamic aggregates more clearly).

3.3. Selecting Properties to be Visualized

Next the user selects attributes to visualize. In this scenario, the user is a real estate agent who is exploring a hunch that many people have been stretching their finances to buy near a city park. Figure 1 also shows the process of selecting attributes from a menu. In this case the agent wants to view house locations, and the relationship between their cost and the ability of the buyers to afford them. Consulting the schema, this requires latitude, longitude, and neighborhood of the houses, selling price of the sales, and salary of the buyer. To visualize these attributes, she wants to create a map showing the first three attributes, and a chart showing the remaining two. To create the map, a SageBrush [10] frame is dragged into the VQE frame (Figure 2 right). The agent constructs a sketch of the desired visualization by dragging a map icon into the work area, and drags copies of the latitude and longitude attributes of the house dynamic aggregate into it. The neighborhood attribute is also dragged to the color property of the point icon on the map. The Create Picture button sends a design request to the Sage expert system, which returns a visualization design incorporating the attributes, in this case a map of the houses (Figure 2 left). In the absence of a sketch, or with a partial sketch, Sage uses heuristics to design pictures to facilitate specified analysis tasks on specified attributes.

3.4. Coordination Across Distinct Queries

The agent then sketches a plot chart in a manner similar to the way the map was sketched. The chart is designed to show the relationship between the selling-price of houses and the salaries of the buyers. The updated state of VQE is shown in Figure 3. In the chart, there is a clear correlation

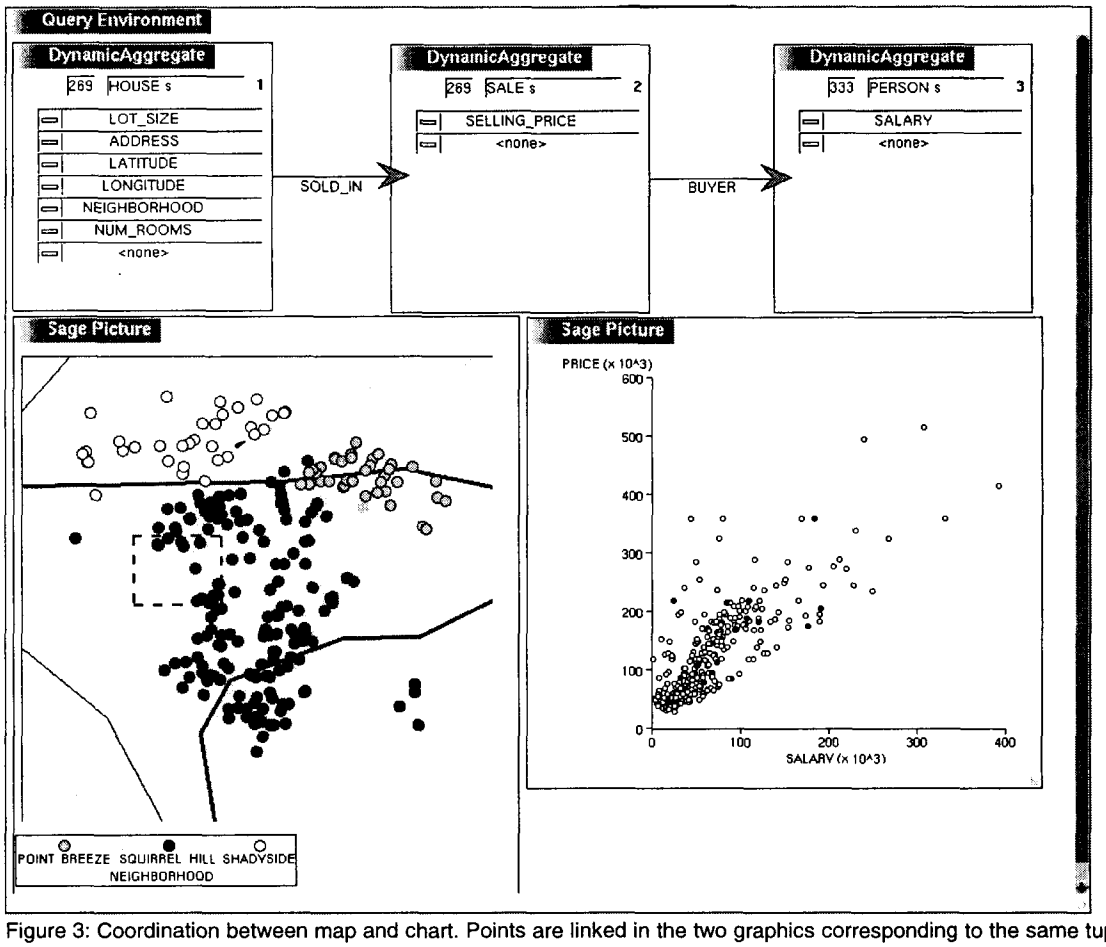

Figure 3: Coordination between map and chart. Points are linked in the two graphics corresponding to the same tuple of house, sale, and buyer. Here the agent has painted the region on the map using a bounding box where she suspects a relationship to the salary-price graph. On the chart, the black points are scattered throughout the distribution, indicating no clear pattern.

between price and salary, but the dependence on location can't be seen statically because there is nothing to visually link corresponding points in the chart and map. The agent then selects the region she is interested in on the map with a bounding box to paint those houses black (Figure 3). The displays are dynamically linked [3], so the points related to these houses are also painted black in the chart. Note that the attributes of different objects are displayed in separate graphics but are coordinated by painting.

No real pattern is apparent in the chart. Recall that the goal of the analysis was to determine locations in the city where buyers are purchasing houses with prices disproportionately higher than others relative to their salaries. Not being certain of the locations where stretching is suspected, the agent would like to go the other way - to isolate the houses with high price/salary ratios, and see where they fall on the map. One way to do this is to use painting again. To highlight tuples where the ratio is above 2.5, for instance, all points above a diagonal line would have to be painted either individually or with numerous small (axis-aligned) bounding boxes. Likewise, to change the ratio of interest (e.g. to 2 or 4), the same laborious painting process would be required. The following section discusses features that make this task easier.

3.5. Definition of New Properties

An alternative is to declaratively specify a new attribute, the price/salary ratio. On the Sales dynamic aggregate, the agent selects the menu item "Create New Attribute". She can then type in a formula as in spreadsheet programs. Here she first drops a copy of the SELLING-PRICE attribute on the cell, types "/sum()", and then drops a copy of the SALARY attribute from the PERSON dynamic aggregate (Figure 4). She names the new property "StretchRatio". The system then creates the new attribute and updates all instances of SALE with the derived value. This is another case in which attributes of distinct objects are integrated.

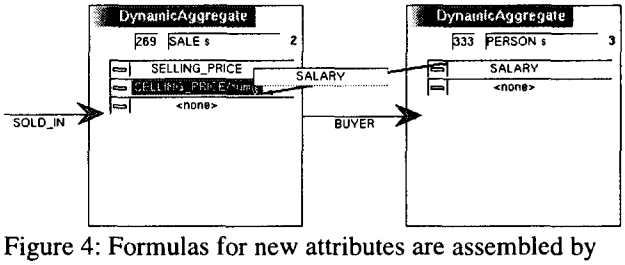

Figure 4: Formulas for new attributes are assembled by dragging references to other attributes, and by typing arithmetic or aggregation operators.

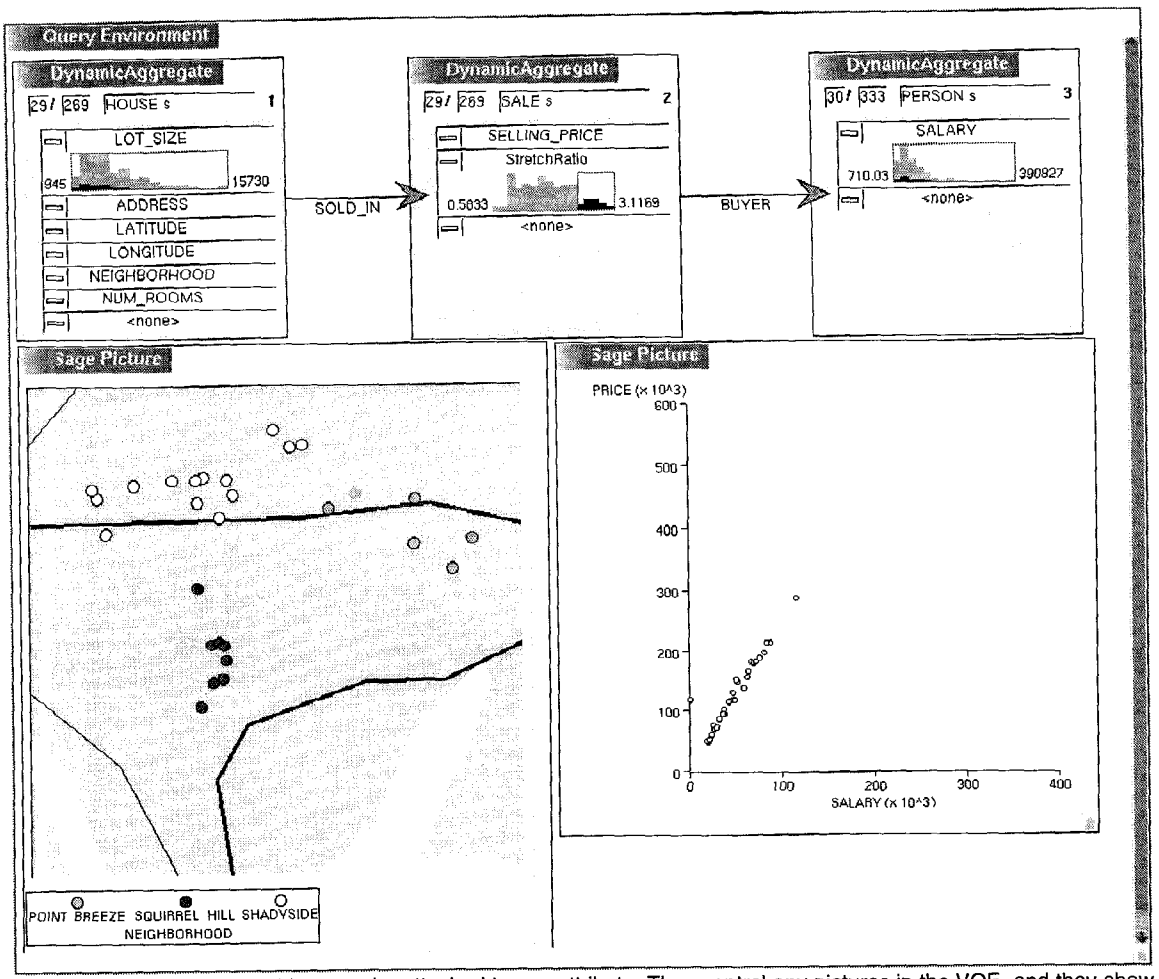

Figure 5: Dynamic Query sliders can be attached to any attribute. They control any pictures in the VQE, and they show a histogram of values for that attribute.

3.6 Filtering Objects

Although it is possible to create a new visualization containing this derived attribute, it is simpler to just dynamically filter objects in the map using this attribute. So instead the agent drops a Dynamic Query slider [2] on the attribute. The interior of the slider shows a histogram, and the border can be moved to filter sales based on the value. Filtering affects the gray-scale of histogram values as in the Influence Explorer [13], as well as the objects in the existing visualizations.

The number of unfiltered entities in each dynamic aggregate is also shown ("29/269 SALE"). In Figure 5, the agent has found a range of very high price/salary ratios for houses near the park. It is also interesting that these represent lower priced houses purchased by people at the lower end of the salary range. These observations are possible because houses on the map and points on the chart are being filtered based on properties of the sales and of the people who bought them. The agent has also dropped histogram sliders on the lot-size and salary attributes. They are not being used to restrict the selection set, but they display the conditional distribution of values for the filtered set (black), as well as the original distribution (gray).

Figure 6 shows an SQL query that would return the same houses found by the exploration process above. It is less transparent than the visual equivalent, and it would be difficult to arrive at the cutoff without interactive, incrementally modified visualizations.

3.7. Contributions

3.7.1. Query Language

This example scenario illustrates iterating between querying and visualization during data exploration, the power that results from integrating these functionalities within a single interface, as well as the power of coordination across

```
SELECT longitude, latitude, salary,
selling-price
  FROM person, house, sale
  WHERE person.id = buyer
      AND house.id = house-sold
      AND selling-price/salary > 2.96
```

Figure 6: SQL query for the houses found above. We have assumed keys for the tables that were not needed in the conceptual level interface.

multiple iterations that results from the shared access to the database objects. The queries required more expressive power than is available in previous visualization systems, yet were simple to construct by virtue of the analogy to direct manipulation operations in Visage. Integration of multiple objects emerges from the navigation process, selecting properties to be visualized is done with a sketch, filtering is done with sliders and painting, and arithmetic operations are entered as in spreadsheet formulas. In previous direct manipulation systems there is no way to pair up all the people with their houses in order to compare salaries to prices. Even filtering out the low price/salary datapoints is awkward because there is no representation of this abstract intent; rather individual points must be painted.

The fact that the query language is visual has several advantages. Quoting [15], Weiland and Shneiderman [14] list 5 problems with textual query languages:

1. Excessive reliance on user recall of names, data formats, and units.
2. Data models that require users to describe implicit relationships via explicit queries (e.g., joins in relational systems).
3. Lack of feedback in the process of formulating queries.
4. No facility for suppressing unnecessary detail.
5. Lack of overview or browsing facility for meta-data (the schema).

Their Graphical Query System addresses points 1, 3, and 4. We have addressed the remaining points (2 and 5) as well. The ER diagram provides a meta-data browsing facility; the navigation and attribute menus relieve the need to recall names; the graph structure and sketch provides feedback on the state of the query; and it uses a more intuitive object-oriented data model, which reduces the amount of low level detail.

3.7.2. Integration via Dynamic Aggregates

In previous query languages, query objects exist at the meta level, and do not correspond to ordinary data objects. VQE bridges this gap by considering the query graph nodes as serving two functions: as *prototype objects* (i.e. variables) and *dynamic aggregates*.

When serving their role as variables, query nodes are bound to particular data objects such that the constraints imposed among the nodes by the relationship links are respected. Each tuple of bindings over all the query nodes can contribute one graphical object to a visualization. This corresponds to the join operation in relational databases, and each tuple of bindings corresponds to a row in the resulting table.

In their role as dynamic aggregates, query nodes collect the set of objects in one *column* of a relational table. If a query node is dragged into a Visage frame, it represents an aggregate of all those data objects that could be consistently bound to the node.

4. Related Work

4.1. GQL

GQL [8] is a conceptual level visual query interface with the expressiveness of SQL. VQE's visual representation of the query graph is adapted from GQL, with some interface modifications such as using containment to show attributes rather than links. However GQL is not integrated with a visualization system for displaying query results. Each query generates a static table, so the paradigm is batch.

4.2. Exbase

Exbase [5] is perhaps the closest existing system to this work in terms of motivation, in that it seeks to provide an intermediary between a database and a visualization system. However the emphasis is on explicitly representing the history of user interaction with the database and visualizations. Lee and Grinstein distinguish remote database accesses from local processing, so there are objects for database views, which are the result of queries, and derived views, for example as the result of manipulating sliders. Similarly, they maintain a derivation history of visualization views.

We have not yet addressed the question of maintaining histories, having chosen to focus on intuitive query languages and integration of querying and visualization -- topics that Exbase, which uses SQL as a query language, has not yet addressed.

4.3. Demonstrational Interfaces

There are two approaches to integrating abstraction into a direct manipulation interface. Demonstrational interfaces attempt to retain the direct mapping from domain objects to visual objects. The meaning is that all domain objects *similar* to the one being manipulated are affected. The problem then is that you can't be vague when manipulating domain objects. Every person has a height, for instance, so if you want to retrieve all people irrespective of height you might manipulate the visual representation of people with varying heights, so the system can induce that for you height is just an accidental property [7].

Other systems, including VQE and SageBrush, manipulate visual objects that are *prototypes* of domain objects. Although this stretches the direct manipulation paradigm, the ambiguity problem does not come up. Although any house has a definite number of rooms, the interface allows an explicit visual "variable" representing it, which can be partially or totally constrained with a slider.

Still other systems use a combination of induction over domain-level manipulations together with an explicit declarative visual language [6]. In the future, we would like to follow this path and induce VQE queries during Visage exploration. The correspondence between action and query must be clear, so that the user can reuse and modify it.

4.5. Large Databases

The rapid feedback in the direct manipulation paradigm is incompatible with querying to secondary storage or remote servers, which will be necessary for large databases. Query Previews [4] offers an interface for previewing data before downloading relevant subsets. IDEA [12] uses interactive manipulation of a *random* subset of the data to form a good query, which is then applied in batch mode to the whole dataset. GUIDE [15] offers suggestions for dealing with large schemas. These ideas could all be applied to VQE.

5. Summary

VQE combines a GQL-style visual language for constructing datasets with direct-manipulation data exploration capabilities as found in systems like Visage, IVEE, and the Influence Explorer. Since queries and visualizations share an object-oriented database, visualizations resulting from a sequence of queries are coordinated. Integration of navigation and query style exploration allows use of direct manipulation where possible but still retains the ability to capture and reuse query sequences as declarative structures. Any frame containing a sequence of nodes and links and associated visualizations can be saved and/or cleared of data to be reused with new datasets.

In the future we want to tighten the integration of the direct manipulation and query paradigms. On the one hand, we want to move navigation closer to querying with heuristics to infer queries from navigation so that any exploration session can be browsed, modified, and reused. On the other, we want to increase the expressive power of the query language to include disjunction, negation, and grouping, which can presently only be expressed by navigation.

Acknowledgements

This project was supported by DARPA, contracts DAA-1593K0005 and N660061-96-C-8503. We are grateful to the members of the Visualization and Intelligent Interfaces Group at CMU, and the Visage team at Maya Design Group for numerous discussions and for valuable comments on this paper.

References

[1] Christopher Ahlberg and Erik Wistrand. IVEE: The information visualization & exploration environment. In *Proceedings of IEEE Information Visualization Symposium, InfoVis'95*. IEEE, 1995.

[2] Christopher Ahlberg, Christopher Williamson, and Ben Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '92)*, pages 619--626. ACM Press, 1992.

[3] Richard A. Becker and William S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2), 1987.

[4] Khoa Doan, Catherine Plaisant, and Ben Shneiderman. Query previews in networked information systems. Technical Report CAR-TR-788, University of Maryland, Department of Computer Science, Human-Computer Interaction Laboratory and Institute for Systems Research, University of Maryland, College Park, MD 20742, 1995.

[5] John Peter Lee and Georges G. Grinstein. Describing visual interactions to the database: closing the loop between user and data. In *Proceedings of Visual Data Exploration and Analysis III (SPIE '96)*, 1996.

[6] Francesmary Modugno. Extending end-user programming in a visual shell with programming by demonstration and graphical language techniques. Technical Report CMU-CS-95-130, Carnegie Mellon, Computer Science Department, 1995. PhD Thesis.

[7] Brad A. Myers, Jade Goldstein, and Matthew A. Goldberg. Creating charts by demonstration. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '94)*, pages 106--111. ACM/SIGCHI, 1994.

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

[9] Steven F. Roth, Mei C. Chuah, Stephan Kerpedjiev, John A. Kolojejchick, and Peter Lucas. Towards an information visualization workspace: Combining multiple means of expression. *Human-Computer Interaction*, in press, 1997.

[10] Steven F. Roth, John Kolojejchick, Joe Mattis, and Jade Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '94)*, pages 112--117, 1994.

[11] Steven F. Roth, Peter Lucas, Jeffrey A. Senn, Cristina C. Gomberg, Michael B. Burks, Philip J. Stroffolino, John A. Kolojejchick, and Carolyn Dunmire. Visage: A user interface environment for exploring information. In *Proceedings of Information Visualization*, pages 3--12. IEEE, 1996.

[12] Peter G. Selfridge, Devesh Srivastava, and Lynn O. Wilson. Idea: Interactive data exploration and analysis. In *Proceedings of SIGMOD 1996*, 1996.

[13] Lisa Tweedie, Robert Spence, Huw Dawkes, and Hua Su. Externalising abstract mathematical models. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '96)*, pages 406--412. ACM/SIGCHI, 1996.

[14] William J. Weiland and Ben Shneiderman. A graphical query interface based on aggregation/generalization hierarchies. Technical Report CAR-TR-562, University of Maryland, Department of Computer Science, Human-Computer Interaction Laboratory and Institute for Systems Research, University of Maryland, College Park, MD 20742, 1992.

[15] Harry K. T. Wong and Ivy Kuo. Guide: Graphical user interface for database exploration. In Proceedings of the 8th VLDB Conference, 1982.