CHAPTER 3

A SYSTEMS MODEL FOR DATABASE EXPLORATION

One way to concretize many issues surrounding database exploration is to focus on its data-centric, i.e., systems aspects. To that end, this chapter conducts a user-centered analysis of two primary data exploration tasks, and applies the analysis to a system architecture that supports those tasks. Such a user-centered systems approach addresses many fundamental architectural issues, most notably parameterizing the "impedance mismatch" between database and visualization systems. Perhaps more importantly, the data-centric result (the software architecture) raises several fundamental issues concerning the nature of database exploration, motivating further research.

Based on Table 2-1, two tasks are central to database exploration: *querying* data to generate data subsets for analysis, and *displaying* (or visualizing) data for examination by the data analyst. This chapter first defines a database exploration scenario that is used throughout the thesis, and a *task repertoire* that is built on these two tasks. An analysis of database and visualization systems for their data exploration capabilities in the context of the scenario is conducted. Then, a detailed description of the *Exbase* system is given, using the analysis to support database exploration tasks. Based on this task-oriented, systems-level research activity, we set the stage for the remainder of this thesis by motivating the need for a user-centered process model of data exploration.

3.1 A Data Exploration Scenario

Database exploration is defined in Section 2.1 as a highly interactive process with a strong visual component that orchestrates the progress of the exploration session. The data analyst generally selects data from a database or data warehouse, and analyzes the selected data using data mining tools (clustering, regression, rule extraction, etc.) and visualization. In this scenario, there are no data mining tools; all knowledge extraction is performed directly by the analyst viewing visualizations of database queries. Since data mining tools must present their results visually, they are just another (highly advanced) analysis tool available to the data analyst. Such tools could be integrated into database-visualization system, however.

A data exploration *task repertoire* is a collection of techniques that support specific kinds of data exploration tasks. This database exploration scenario focuses on database and visualization data domains. The database domain encompasses database management and selection (i.e., querying). The data visualization domain encompasses data presentation for visual analysis, and support for data interactions (i.e., the user interface to data).

The task repertoire of our database exploration scenario includes:

1. queries must be issued to the DBMS
2. query results must be visualized
3. query results must be queried
4. visualizations must be queried

We now elaborate on each fundamental task.

1. *Queries must be issued to the DBMS* - The data analyst must first issue database queries, using some query language, to select and reduce data for further investigation. Data reduction is a critical operation for large databases, and is best handled by the DBMS, supported by query tools. In the visualization pipeline model of Figure 2-7(a), the data enrichment and enhancement step is now represented by a database query. This elevates data selection from an arbitrary operation to one that has a known expressive power, based on the query language and data model used.

2. *Query results must be visualized* -Traditionally, a textual data display is used by database systems. For large query results, graphical methods are appropriate, even essential, for the data to be comprehended by the data analyst.

3. *Query results must be queried* - At times, the analyst may wish to query data that has already been retrieved. This could be to isolate a smaller region of the result, or to compose a larger result from component pieces.

4. *Visualizations must be queried* - The visualizations of query results mark decision points for the data analyst. If a visualization is interesting, the analyst might want to isolate a portion of it for in-depth study. Querying visualizations might involve selecting data within a region of interest in the display (by graphical panning or zooming), reconfiguring the data displayed (by a graphical projection or viewing operation), or by changing the graphical display parameters (lighting, colormap, etc.). Thus, a query in terms of a visualization can be broadly defined to include all image-altering operations.

The essence of database exploration is the repeated execution of database and visualization tasks. Based on visualizations, the analyst decides whether to query for data, alter the visualization, or perform some other interaction. Interactions can be targeted at

the current visualization or its underlying query result, the database or some intermediate query result or visualization, provided they are retained by the database exploration system.

Thus, in our database exploration scenario, the data analyst is restricted to performing querying or visualization operations, but is free to revisit previous query results and visualizations, and perform query and visualization operations on them as well. This notion corresponds to similar ideas found in the literature that further interactions are spawned from existing analyses and visualizations of queries (Buja 1996; Nicholson 1984; Owen 1986; Velleman 1990; Young 1991b).

3.2 A User-Centered Analysis of System Components

This section takes a closer look at the database and visualization components of a database exploration system, by analyzing their intrinsic data models, data access paradigms, and data interaction capabilities. As a result, this analysis outlines the primary database exploration systems issues. It is important to note that data interaction capabilities play a critical role in developing these issues, because they are directly affected by the task repertoire.

3.2.1 The DBMS Component

The DBMS component under consideration is the relational database management system. Relational database management systems are widely used for on-line transaction processing (OLTP). They have many advantages: they manage persistent tabular base data and provide selective, concurrent access for data insertion, retrieval, and updating operations. They excel at identifying and selecting primitive base data (simple numeric, text, decimal and date types) based on some qualification. This makes them suitable for

reducing both the complexity and volume of data. Extended relational and object-oriented database management systems manage more complex data types.

Relational databases are suitable for database exploration, because they are built on a simple yet formal data model, possess a non-procedural data manipulation language, and support persistence, integrity constraints and concurrency. Corporate relational databases are interesting for data exploration because they can be very large (on the order of terabytes), and of high dimensionality (hundreds or thousands of data attributes). There is also a great need to extract useful knowledge from such databases within the business community. The lack of any intrinsic data spatiality (aside from, perhaps, a geographic location associated with an address) creates interesting visualization challenges that are specifically addressed by our visualization component.

3.2.1.1 Relational Data Model and Access Paradigms

The relational data model is a formal model based on a single data structure, the *relation*. A relation is essentially a table of values. Each row (called a *tuple*) is a collection of related data values. Each column signifies an attribute, which has a specific data type. Precise descriptions (using the relational algebra) can be found in any standard database system textbook. Database access is through the structured query language (SQL), a non-procedural data manipulation language. Conceptually, a query maps the entire database onto a single relation. For data exploration, the SQL query is the fundamental operation for data retrieval, and has the following structure (square brackets are optional components):

```
SELECT          < attribute - list >
FROM            < table - list >
[WHERE          < condition >]
[GROUP BY       < grouping attribute(s) >]
[HAVING < group condition >]
```

[ORDER BY < attribute - list >]

The SELECT clause identifies the attributes to be retrieved or functions to be applied. Functions are aggregate functions such as COUNT, SUM, MIN, MAX, and AVG. The FROM clause identifies the all of the relations required by the query. The WHERE clause specifies the conditions for selecting the tuples from the relations. Conditions often involve comparisons such as >, <, >=, <=, =, and !=. In some instances, aggregate functions can be applied on specific subgroups of tuples, so the GROUP BY clause specifies the groups to be materialized. The HAVING clause specifies a condition to be enforced over the groups of tuples associated with each value of the grouping attributes. The ORDER BY clause specifies which attributes can be used to order the resultant tuples. Thus, the basic SQL query gives a number of important functionalities to the data analyst in selecting data of interest.

Query results are relations, but not always sets because they may contain duplicate tuples. The DISTINCT keyword eliminates these duplicate tuples. SQL provides a number of additional operations for substring comparisons, arithmetic operations on numeric values and set operations. Set operations include UNION, EXCEPT (set difference) and INTERSECT. Duplicate tuples are removed during these set operations. Queries can be nested, where an inner query is embedded within the WHERE clause of an outer query. The inner SELECT fetches tuples for further evaluation by the outer query.

The *view* mechanism allows a name to be associated with a query, and that name used in other relational expressions. This simplifies querying, and lends object identity to relational processing. Views can be queried, just like base relations. They may be *materialized*, constructed directly by the DBMS when first specified, or they may not be materialized, and constructed each time they are referenced.

3.2.1.2 Interaction with Relational Databases

Data interaction in relational databases is exclusively through SQL, which allows the selection of relational data satisfying a certain condition. It also offers primitive computations on database attributes, and some reorganization of the tuples making up the query result. SQL also provides a limited form of navigation of the query result through a *cursor*, a pointer into the current tuple retrieved from the database. The cursor can be incremented or decremented. The traditional method of data interaction for relational databases is to issue a textual SQL query at a client terminal, and receive a cursor into the textual query result stored at the server.

The DBMS normally does not provide query result management, so this functionality must be implemented by a client application that understands the database structures and can accept the cursor. Using either an application programmers interface (API) or embedded SQL, the client application can continually fetch a row from the cursor, convert each member of the character buffer into an atomic data value of the appropriate type, and build an application data object.

In order to make database interaction more efficient, additional front-end tools can be implemented at the client. Most commercial database systems and many third party vendors offer such tools. Basic tools include browsers to view the database schema (the collection of base tables), querying tools using menus, forms and graphical construction techniques, and report generators that can organize and display the result set in a tabular form, or as simple graphics (bar, line and pie charts, scatterplots). Query result data can be fed into spreadsheet applications and also be used in subsequent query construction. More advanced interaction tools provide functions to manipulate the data display, such as

changing the display axes, navigating hierarchies for more detailed or summary data, and applying financial data analysis functions.

The style of interaction with relational databases depends upon the application. For traditional OLTP applications, there are usually numerous clients accessing the database via small, precise queries that return small results. For analytic operations common to on line analytic processing (OLAP) applications, a multidimensional database system accesses the relational database tables, returning larger amounts of base data or preprocessed summary data. For scientific database-type applications, which would serve most visualization systems, the queries would also retrieve large amounts of data, possibly with analytic functions specified within the queries, to be executed at the database server.

3.2.2 The Visualization Component

Data visualization systems focus on graphical data presentation and analysis in diverse domains such as physics, biology, finance, mathematics and applied engineering. Lately, the field of “information visualization”, or the visualization of nonspatial data, such as text documents, has become popular. Modular Visualization Environments (MVEs) are popular systems for constructing visualizations that contain a library of functions (modules) that can be configured using a visual programming environment (VPE) to construct a visualization application in the manner of the GuideMaps and WorkMaps of Figure 2-16. Links between modules are the dataflow paths that are predominantly unidirectional from data sources to images, mimicking the traditional visualization pipeline of Figure 2-7(a).

Iconographic data visualization is a technique suitable for data represented as discrete points in multidimensional data spaces, such as database relations, where tuples

are equivalent to multidimensional data points. This technique maps the attributes of each data point in a consistent fashion to the geometric features of each graphical icon. Figure 3-1 shows an example of a mapping between a database relation and a line-oriented icon. Two attributes are mapped to the *(x,y)* position of the icon, and the remaining five attributes are mapped to the angles between pairs of adjacent line segments of the icon.

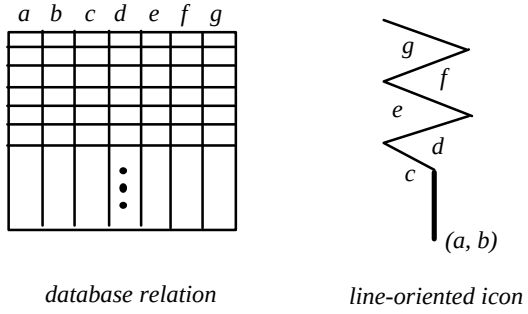

Figure 3-1. The mapping between database relation and a line-oriented icon.

One prominent display method that is the basis for the *Exvis* exploration visualization system densely packs these line-oriented icons into a two-dimensional array, producing a visual texture that is discriminable by the data analyst (Pickett and Grinstein 1988; Grinstein, Pickett and Williams 1989; Smith, Grinstein and Pickett 1991; Grinstein et al. 1992). Figure 3-2 shows an Exvis iconographic display for two merged MRI brain scans of a tumorous area. Each data point is a tuple of (*x, y, v1, v2*) where *v1* and *v2* are the MRI scan values at each (*x,y*) position for scan 1 and scan 2, respectively.

In Figure 3-2, the iconographic technique produces a rich, structure-laden visual texture. The tumor is clearly evident in the upper-left portion of the image, and the tumor "hot spot" is also apparent. The skull cross-section is visible across the bottom of the image. The basis of the Exvis approach lies within the preattentive line-processing capability of the human visual system. Portraying data in this manner may aid the analyst in

rapidly assessing embedded data relationships, which is an essential component of database exploration.

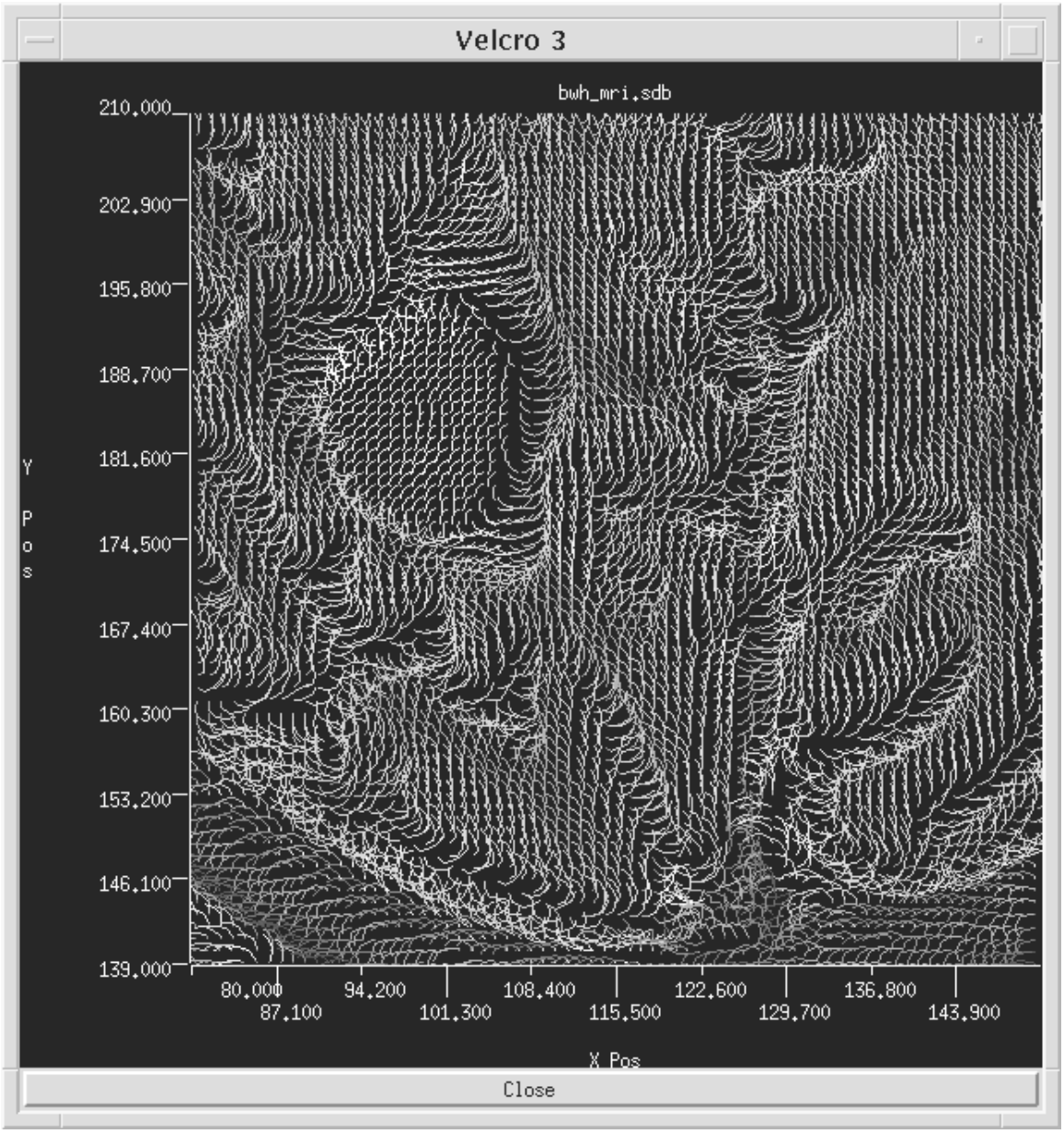

Figure 3-2. An Exvis visualization of two MRI scans.

The iconographic display technique uses simple computer graphics and is easily implemented. Furthermore, it creates visually compelling displays that shows relations as graphical structures within texture arrays. This technique could be implemented as a module within a MVE.

3.2.2.1 Visualization Data Model and Access Paradigms

In general, visualization deals with “some other data”, meaning that the underlying data model is that of the data that must be visualized. Often, the data is transformed from its native format into a data structure that lends itself to efficient graphical processing (such as display lists, quadtrees and octrees).

Data for visualization is most often considered read-only, so MVEs lack database concurrency, integrity and security overhead. More importantly, they also lack associative data access. Image-space and object-space data structures such as quadtrees and octrees do support spatial queries, but are not widely used in practice. Data models for visualization are an active research topic: there are no formal standard models because of the diversity of data types. Active areas under research are fiber bundles and multidimensional lattices. Multiple data models might need to be created that are domain- or data- dependent.

Commercial visualization system data models are proprietary, though there are public domain data interchange formats (such as CDF, netCDF and HDF) that are used as underlying data models for some environments. They too have limited data access capabilities, however, usually limited to bulk transfers, or perhaps regular sampling based on position. A common approach is to model data as a *field*, having data, positions and connectivity components. This field structure is self-describing, making data processing

flexible. Some commercial visualization environments are just beginning to offer access to data stored in relational databases.

There are no formal data manipulation languages for visualization environments. Scripting languages and APIs do exist however, to configure and execute dataflow networks. These languages often offer a superset of the functions afforded by the graphical user interface. Data manipulation capabilities, including data access, are usually associated with each particular module.

3.2.2.2 Interaction in Visualization Systems

MVEs offer a high degree of configurability. They permit data interaction with the visualization process (network construction), visualization module parameters, and the visualized images. Data interaction with module parameters is generally realized through graphical widgets (dials, sliders, and text / numeric fields) that are part of the module network. These *control widgets* control simulation or visualization parameters, and can be either user-driven or data-driven. The numerous types of control widgets include scalar, vector, string, list, selector and file selector controls, colormap editors, animation sequencers, view control, spatial navigation controls, graphical selection controls for probes and picking, rendering options and orientation aids such as spatial axes and bounding boxes. Interaction with visualized images includes pick-and-querying points on the image, reorienting the spatial view and specifying a rectangular region of interest in the image for further processing.

Visualization functions include 2D and 3D representations of scalar and vector data using points, lines, surfaces and volumes. These can be enhanced with color and opacity maps, and animation for temporal sequences. Visualizations may be “queried”

using cutting planes within volumes, object picking to select objects within an image, and data probing to determine the location of a value within an image. Analytic functions include spatial property calculations (length, area, volume), univariate statistics (min, max, mean, standard deviation), pointwise mathematical expressions (arithmetic, transcendental functions, etc.), and image processing functions. Data manipulation capabilities include removing data points, subsetting by position, subsampling, supersampling, grid construction, interpolation, display axes transposition, etc.

3.2.3 System Requirements for Database Exploration

By harnessing the strengths of database and visualization systems, we can realize a powerful integrated database exploration system. Relational database systems have formal data models, well-defined query languages and provisions for concurrency, reliability, scalability and security, making them a desirable foundation. Modular visualization environments enable huge volumes of complex data to be presented in a comprehensible format, and provide many graphical tools for manipulating the visualization. Perceptually motivated visualization techniques, such as the iconographic approach taken by Exvis, hold promise for more effective presentations.

The database exploration scenario, however, places conflicting requirements on the system architecture. Not only must the architecture support associative data access (implying a complex implementation of a database model), it must also support visualization (possibly implying a radically different type of data model) and visual querying (implying a complex set of data manipulation operations). This impedance mismatch is perhaps the most critical challenge facing database exploration architectures. The visualization component must be able to manage data retrieved via a cursor, and

adapt to extremely large query results. Normally, visualization systems make bulk data transfers from files, which is not the typical database system approach.

This analysis can be summarized with the following minimal set of requirements for any integrated database exploration system. These requirements can serve as a "litmus test" for evaluating any proposed database exploration system. The integrated system must

1. *Maintain query results for further exploration*. This is the most basic requirement. Building retained collections of query result tuples supports the central database exploration concept of enabling the analyst to manipulate numerous data objects. Each query result represents some relationship among the data, that may be useful to the analyst.

2. *Provide techniques for visualizing high-dimensional data sets*. This requirement maps the database data model to the visualization technique (and associated data model). Since the visualization technique must be able to display high-dimensional data points (i.e., a tuple), the iconographic approach is particularly useful.

3. *Permit associative data access over numerous types of data objects*. This requirement maps the visualization's interaction model to the DBMS. Potential queryable data objects include database tables, intermediate data objects such as query results and materialized views, and also visualizations themselves. Visual query tools must be implemented, such as multidimensional data probes and brushing techniques, which are main-memory generalizations of the traditional database cursor.

4. *Support rapid sequences of user-data interactions*. Though not entirely apparent from the analysis, this requirement stems more from the nature of database

exploration presented in Chapter 2. Exploration implies a “fast and loose” data manipulation scheme, where the analyst effortlessly manipulates data in a rapid fashion. Most interactive visualization techniques stress high performance, requiring specialized graphics hardware. The utility of direct manipulation places enormous burdens on interactivity, so design decisions must carefully balance capabilities with performance.

These requirements suggest an integration strategy (cf. Section 2.3.3) that is part *visualization as a database front end* (Requirement 2), and *database as a visualization back end* (Requirement 3), because displaying and querying are tightly integrated to satisfy Requirement 4. In the following section, we describe a research prototype software architecture that was designed to address each of these requirements.

3.3 Exbase: An Integrated Database Exploration Environment

In order to meet the varied requirements of database exploration, we have developed the *Exbase* research prototype. Exbase is an object-oriented interface between a database system and visualization system (Lee 1994; Lee and Grinstein 1995; Lee 1996), and can be considered an evolution of the Exvis exploratory visualization system to accommodate relational databases and enhanced user-data interactions at the visualization display.

In this section, we give an overview of Exbase and place it in the context of the categorization of database exploration systems defined in Section 2.3.3, we describe how the integration is accomplished in terms of database structures and visualization structures, and we finally show how user-data interactions are accomplished.

3.3.1 Exbase Overview

The Exbase system resides within a database client visualization application, and is thus tightly integrated with the visualization system. Exbase brings database and visualization closer together by managing queries, query results, visualizations, and maintaining sequences of these data objects within the database client, as shown in Figure 3-3. Exbase forwards queries to the DBMS that are generated at the visualization, and maintains their results. Exbase accepts visualization specifications from the visualization system, and supplies data for visualization to the visualization system.

The core concepts of the Exbase architecture are that it

1. supports the database data model

2. offers database interactions at the visualization display, to "bridge the gap" between database and visualization systems, (cf. Figure 2-16)

Exbase uses the relational database data model, since the primary focus is to explore relational data. In doing so, Exbase directly supports the database exploration tasks described in Section 3.1.

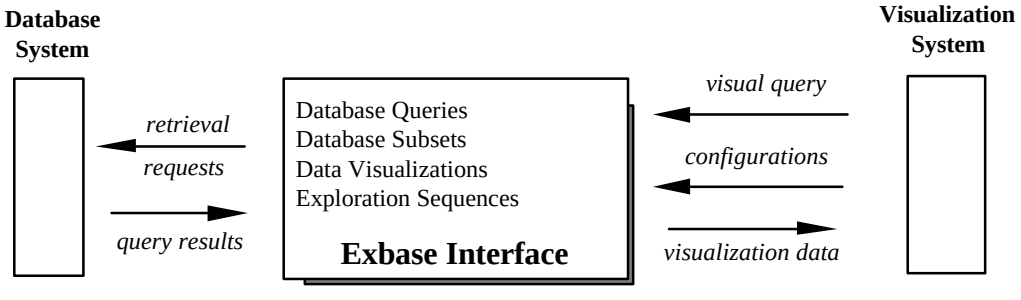

Figure 3-3. The Exbase Interface, with primary data objects and operations.

In comparison to the integration strategies presented in Section 2.3.3, Exbase is a hybrid strategy, because it treats database and visualization systems as peers, and not as a master-slave pair. Though the iconographic technique is chosen because it is a natural way to visualize relational data, it also has database operations available at the visualization display, so Exbase can drive the database as well as be driven by it.

3.3.2 Exbase View-Based Integration

In arriving at a unifying integration scheme, we have focused our attention on the concept of a *view*. Views are precisely defined in the database domain but imprecisely defined in the visualization domain. Database views relate to the formal relational data model, visualization views generally refer to the operations involved in setting up the viewing transform for graphical rendering.

The Exbase view integration model relies on two fundamental view classes: local database views and visualization views. Figure 3-4 shows an adaptation of the traditional visualization pipeline of Figure 2-7(a) to data exploration. It shows the view classes, the primary types of data transformations, and the feedback paths used by the analyst to interact with the visualization pipeline.

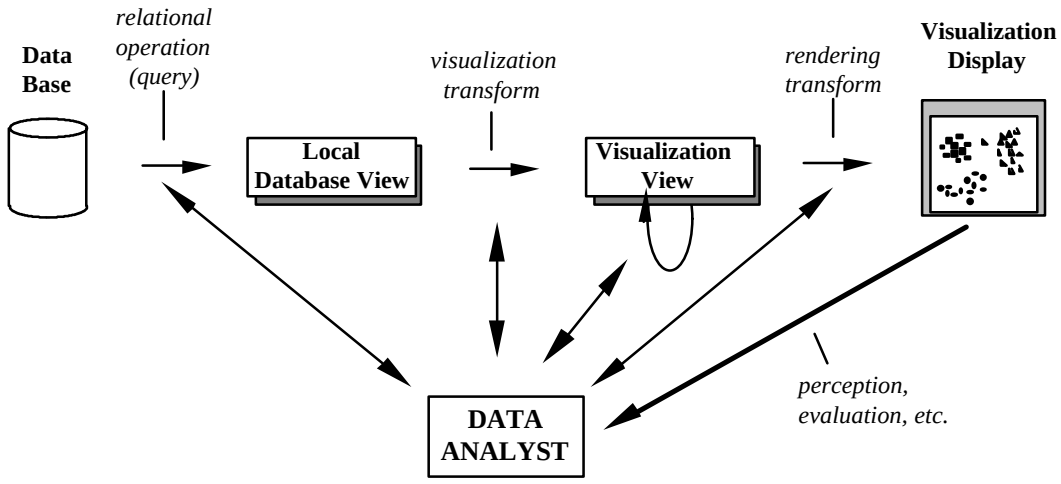

Figure 3-4. Exbase view structures, relevant transformations and interaction paths.

Figure 3-4 implies the analyst interacts primarily with data objects through data transformations. The analyst manipulates the database via DBMS management operations (such as database administration) and data retrieval queries. Queries are data selections, creating local database views. Visualization transformations create visualization views from local database views by mapping data domain structures to display domain structures. Additional visualization-based operations, such as data mappings and viewing transformations, can be applied over the visualization view to create other visualization views. Finally, graphical rendering transforms the visualization view into a data visualization, a graphical image of the extracted and/or analyzed data.

3.3.3 Local Database Views

The local database view is at the heart of the Exbase view integration model, and is derived from the relational database view concept. The relational database view is a temporary or “virtual” relation that is typically not stored in the database, but constructed when it is specified in a query. Database views can be queried themselves, because the

relational model is closed under relational operations (selections, joins, projections, unions, etc.). Exploiting relational closure by allowing successive querying over extracted data is a very powerful mechanism in supporting data exploration. Views are, in general, only in first-normal form, a table of atomic data values.

A local database view, like a relational database view, is specified by three components:

ldb_ view *(name, attribute_list, query),*

where:

name	= the name given to the view,
attribute_list	= the (optional) attribute list of the selection,
query	= the query specifying the view.

The local database view is referred to by its name, and the optional attribute list permits the renaming of view attributes. This definition only specifies the local database view schema; the *extension* of the local database view is the query result. Because the local database view extension resides in memory[1], it can potentially be used to answer subsequent query and visualization requests. It can also be stored in the database if the DBMS permits. Updating base data through local database views is also determined by the DBMS. Client view update facilities would further complicate matters; they are not considered in this research.

3.3.3.1 Relational Operations

The local database view must support traditional database operations such as projection, selection and set operations. It must also support additional constructs and

[1] For the implications of this design decision, please refer to Section 7.4.1

operations not supported by the relational algebra, but supported by commercial database systems for practical purposes.

Projection, Selection and Set Operations

The local database view must offer some relational data operations to the visualization system, because such operations can be specified graphically over the visualization. The two most important operations are projection and selection. Projection chooses attributes and selection chooses rows from the view, as they do from database tables. They can be also be applied to multiple views that are joined together. There are numerous ways to join tables, and some produce results that might not be meaningful. This is an especially critical fact for database exploration, because it is important *not* to create data for visualization that either is not present in the database, or has no meaning in the current context. Furthermore, composing visualizations having unwanted artifacts from incorrect data is unacceptable. It is sufficient to say that local views to be joined should have a common attribute between them, so only valid candidate rows are chosen. Otherwise, an *outer join* operation must be specified, where tuples not having common attributes contain NULL values.

There are two possible types of selection: those that restrict local database views, and those that expand upon them. Restriction is often exemplified by a point query (retrieve rows from a single display location), a range query (retrieve rows based on a range of values of an attribute), or an *ad-hoc* query that possesses a narrower selection criteria. A selection that expands upon a local database view is still considered a restriction of the entire database. Sometimes queries over local views require additional data to be retrieved from the database. For example, in a financial database application, a point query might need to return company identification information along with the actual attribute

values visualized. If the company information is not available in the view, a query must be issued to the database for that data. In this case, a number of rows can be retrieved based on the data driving the current display. Another example would be if a range query is specified that is larger than the range in the view, but partially or fully within the range in the database. An additional query will be required to the database to retrieve that data.

Other meaningful relational operations include set operations (union, intersection and difference) that operate on two local views. As with joining views, there are certain restrictions on performing set operations over local database views. The views in this case must be *union-compatible*, meaning they must have the same attributes in the same order, to produce a meaningful result. As before, it is important not to create artificial relationships within the retrieved data. Exbase supports simple SELECT-FROM-WHERE queries on database tables, and range selections and projection over single local database views.

Additional Constructs and Operations

There are several database system constructs and operations supported by the local database view that are not defined in primitive versions of the relational algebra and calculi: missing values, aggregate functions and query result ordering. Missing values are common in real-world databases, and Exbase missing value handling follows that of the underlying database. They are also accounted for in local database view aggregate functions. When a local database view is created or restricted, the cardinality, number of missing values, and meaningful summary statistics (mean, extents, moments, etc.) are automatically calculated for each attribute, in addition to a sorted index array.

3.3.3.2 Visualization Considerations

The local database view must supply data in an easily visualized format. The interactive nature of database exploration requires rapid responses to the graphical interactions at the visualization display that reorganize the presentation of the local database view. This requirement influences the data format, data access structures and various support functions.

Visualization requires a numeric data representation for display, and statistical data normalization (as opposed to relational data normalization) is a necessary stage in the rendering process. Since many visualizations may be composed for a single local database view, Exbase takes an eager approach to statistical data normalization by creating a normalized floating-point representation of the local database view in addition to the retrieved data. Each distinct local database view is essentially a “world” unto itself, and the extents of all the component data attributes constitute the extent of the world coordinate space of the view. This is similar to the normalized projection coordinate representation common to graphics and visualization.

Some of the many potential visualizations of a local database view require the view to be subsetted in memory by *visual* projection and selection operations. For example, data range restriction is a generalization of the visual zooming operation, a common technique for focusing on an interesting portion of the data. In the visual panning interaction, a data subrange is translated through a larger data range. Each technique causes a different data subset to be displayed. Each technique causes a different subset to be displayed.

To support these tasks in Exbase, the local database view configures a *selection vector* supplied by the visualization view that is used as a mask for visualizing only those rows that satisfy a specified data range. This vector is created for each data restriction interaction instead of copying data to a new local database view. A query string specifying each restriction is also created, in addition to any new statistical metadata on the attribute distributions.

3.3.3.3 Local Database View Implementation

Figure 3-5 shows the primary components of the local database view. The data are stored vertically in arrays on a per-attribute basis, along with metadata such as summary statistics. The additional normalized floating point representation more closely resembles a table, and has sorted indices that point to entire rows based on attribute values. Missing data values are clustered at the end of each index array, and are not used in lookup operations. The base data arrays and index arrays are only manipulated by the local database view. External objects, such as visualization views, use projection and selection vectors to process subsets of the local database view.

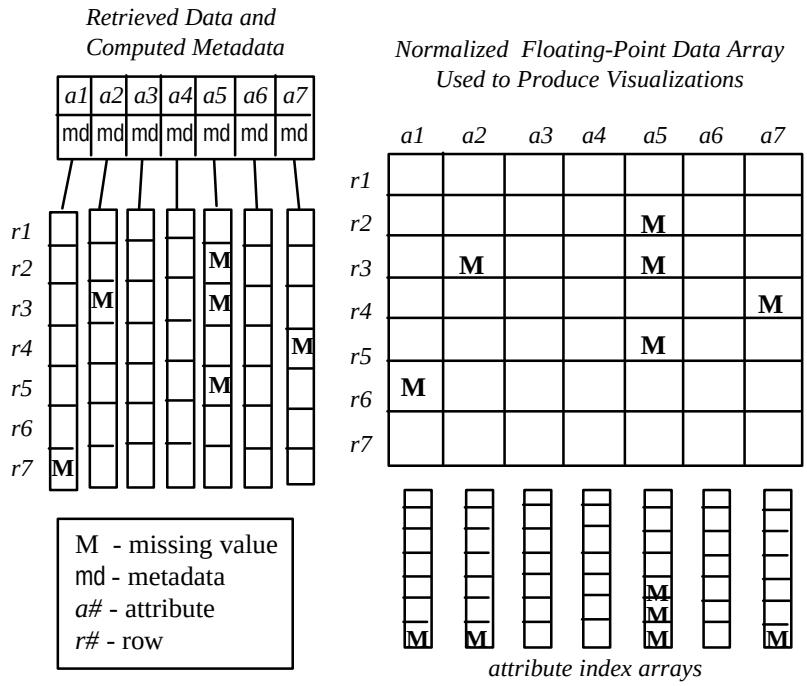

Figure 3-5. The components of the Exbase local database view.

Since the relational model is closed under projection and selection operations, a new object does not need to be created with each new selection or projection, only new access vectors and queries. If we must go back to the database, a new local database view will be created.

3.3.4 Visualization Views

A visualization is a graphical presentation of a data set, i.e., an image, where data values are mapped onto the graphical elements that compose the visualization representation. The visualization process is comprised of two separate transforms. The *visualization transform* maps a set of points in some data space to another set of points in a display space. These data points often have multiple data values associated with them. The *rendering transform* accomplishes the drawing of the data image to the display, using

traditional computer graphics display techniques. Both visualization and rendering are considered data output functions: the visualization transform prepares the data for display and the rendering transform displays the data.

A *visualization view* specifies the visual interface to a local database view. It extends the notion of visualization solely as an output transform by specifying view manipulations in addition to describing the input data and visualization transform.

A visualization view is specified in the following manner:

vview *(ldb_view, ldbv_xfms, vis_spec, int_mappings)*

where:

ldb_view = the underlying local database view,

v_xfms = the sequence of data transforms applied to the database view,

vis_spec = the visualization specification *(vis_rep, mappings, settings):*

- *vis_rep* = visualization representation scheme,
- *mappings* = data-to-visual primitive mappings,
- *settings* = visual display settings,

int_mappings = interaction mappings.

The visualization view is a surrogate to the local database view, because it visually represents the local database view, and because the user primarily interacts with visualizations. Thus, it requires the local database view in it's specification. It also must specify the sequence of possible data transformations that link back to the local database view. The visualization specification is the classical method of describing a data visualization. A representation scheme describes the visual output (scatterplot, isosurface, volumetric, etc.), in terms of its display type, display dimensionality and underlying data

model. Data-to-visual primitive mappings indicate which attributes (columns) of the local database view are represented by the available graphical primitives of the representation scheme. Visual display settings reflect any configurable parameter of the visualization representation and graphical display environment. Finally, the interaction mappings describe the input model over the visualization, and forward graphical articulations to domain data operators as in (Norman 1986).

In typical OLTP database interaction, the visualization is a textual representation that closely matches the stored data representation (a relation) and the mental model the user has of the data (a set of tables). The user often interacts with the database using a textual query language, or through some type of forms interface. In database exploration, the visualization offers two types of visual interactions to explore the data. *Data-independent* (or *data invariant*) interactions are directed towards the visualization display itself, and *data-dependent* interactions are directed at the underlying data (Seetharaman 1994).

3.3.4.1 Data-Independent Visualization Operations

Data-independent visualization operations do not alter the local database view, only the rendering transforms. They should be controllable at the visualization user interface to give the user flexibility in fine-tuning a rendering. Some controls affect graphical structures, such as polygonal or iconic component size and color factors (Grinstein et al. 1992), or the amount of randomness (jitter) applied to icon locations to minimize strong horizontal or vertical artifacts in the display. Other controls affect viewing operations, such as rotating, scaling or translating the data or the display axes. Environment controls affect the general graphics environment variables, such as lighting and the sequencing through animations.

Graphical methods for exploring visualizations have been used in the statistics community since the early 1970s (see (Becker 1988) for an overview). One important manual method is *data brushing*, where several linked (potentially orthogonal) plots of a data space are produced, data is selected in one display using the cursor, and the same selected data in a linked display is highlighted. In this way, multidimensional clusters and other relationships can be determined graphically.

Data brushing reinforces the notion that selection and comparison are fundamental exploration operations deserving support by the visualization along with image generation. Selections are often performed by specifying data subspaces for display or brushing linked displays with the cursor. Brushing plots usually does not require a new subset creation from the local database view. It instead merely changes the appearance of the visualization primitives associated with the same data objects in a linked display.

3.3.4.2 Data-Dependent Operations

The data-dependent operations performed at the visualization include selections and projections to enable the navigation of the multidimensional data space. The visualization view contains a *selection* bit vector that signifies which rows of a local database view are to be rendered. All selections are performed by the local database view object, and require a mapping and rendering to be performed on the new subset. Spatial queries specify some region of display space, and include point queries that isolate a single data object. Range queries specified through a controller can alter one or more attribute ranges at a time.

The visualization view also possesses a *projection* vector that contains attribute identifiers signifying the valid (and possibly redundant) attributes, and the visualization

primitives each attribute is mapped to. This operation is a permutation of the data already available, and a new data subset does not have to be computed. Thus, the effect of this operation is local - only a mapping and rendering of the previous subset are performed. Associated with this vector is a vector of attribute ranges that reflect the ranges of values of each attribute displayed in the visualization.

The visualization process can apply its own local data derivations to database views, that might further restrict or expand the data to be displayed. The visualization transform already alters the data format. Other transformations include sampling the data, logically combining certain attributes, mapping a subset of the available attributes, graphical clipping and interpolation. Also, if two rows of a local database view possess similar values for two attributes, and these attributes are mapped to the two spatial display axes, then they "collide" in the display space. This means that they occupy the same location on the display, and steps must be taken to handle this eventuality. Options include averaging values, cycling through values, adding a jitter offset or overlaying icons at the collision location. Thus, the data displayed, derived from a local or derived database view, may not show the entire view, or may display more data than contained in the view. This affects any subsequent selection operations. Note that these derivations might be able to be pushed back to the derived database view, to simplify the visualization implementation.

Another consideration is supporting missing data. Missing data is common in real-world databases and telemetry, and commercial databases all have a missing data representation. The visualization system component might handle this by not rendering objects containing such data, interpolating new data, or mapping missing data to some sentinel value for display. This should be application dependent, and should be under user control.

3.3.4.3 Visualization View Implementation

The Exbase visualization view is implemented in an object-oriented hierarchy of classes, as shown in Figure 3-6. The **VisualizationView** abstract base class contains a link to the underlying local database view, a query specification (which might differ from that of the local database view, based on the subsetting criteria), a textual description, statistical attribute metadata and a selection bit vector signifying which rows of the local database view are being visualized. The **Visualization** abstract base class contains graphical user interface (GUI) code common to all concrete subclasses (including data selection control widgets), the set of data-to-visual primitive mappings, and the projection vector that defines the visualization mapping.

Concrete visualization classes are subclassed from the **Visualization** class, and implement their specific functions to map, render, and display coordinate axes, among other functions. Concrete classes also implement their particular type of spatial data selection operations. For example, in Figure 3-6, subclasses of the **CartesianVis** class have Cartesian projection display spaces, so their visual displays and region definitions differ from the **ParCoordVis** class, which has a parallel projection display space. New visualization techniques need only supply the unique mapping, display and certain input functions that they possess. Our **VisualizationView** class currently supports simple data derivation transforms, such as regular sampling of data values. Interaction mappings are currently supported by methods of the local database view class that alter the selection vector of a visualization object.

The visualization hierarchy has been constructed to suit relational data. Relational database data often does not have any intrinsic spatial structure, topology or connectivity, as typical “scientific” data does. These qualities are imparted to the data during the

visualization process. Our visualization techniques are iconographic, and do not utilize any structured spatial data representations such as regular grids, though implicitly, the data are stored in a two-dimensional array.

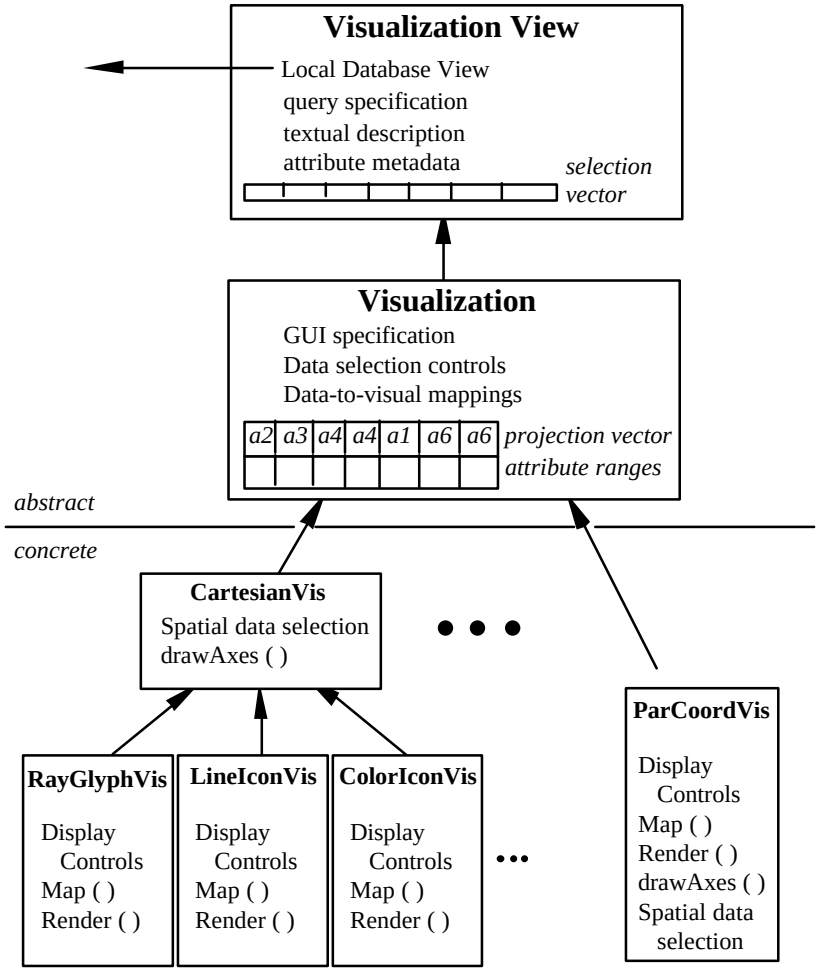

Figure 3-6. Exbase Visualization View hierarchy.

Note that each concrete visualization class provides its own *Render* function, as opposed to having a single, high-level rendering routine implemented. This is done primarily for performance reasons. Having a high-level rendering routine would still necessitate function calls to lower-level classes to render each icon representing a data

point, or use some type of complicated case statement control flow. In this situation, the call overhead, in light of rendering tens or hundreds of thousands of data points, would be unacceptable. Thus, the object-oriented granularity is limited to the visualization technique level. Efficient, if somewhat redundant, rendering routines are then implemented.

3.3.5 Exbase in Action

The following set of figures show screen shots of the main graphical user interfaces of Exbase. The database, annual report data for 3035 natural resources companies over 18 years, is split into two relations, *CompanyInfo* and *FinancialInfo*. Table *CompanyInfo* contains the name and location of each company, and *FinancialInfo* contains the annual report data, 50 attributes per tuple.

Figure 3-7 shows the Exbase textual query interface that contains some usability enhancements, such as selection lists for relations and attributes, ranges of acceptable attribute values (a form of metadata), and save/recall query buttons. The interface makes use of two attribute identifiers, a long English description (such as cost of goods sold) and a concise identifier (cogsold). The query used in this example is:

```
SELECT    r_d_sales, net_assets, t_assets, cogsold, sga_exp, mkt_val_eq
FROM      FinancialInfo
WHERE     FYdate BETWEEN ‘1-Jan-1980’ AND ‘1-Jan-1981’ AND
          r_d_sales BETWEEN 0.01 AND 100 .0 AND
          t_assets BETWEEN 0.0 AND 1000.0
```

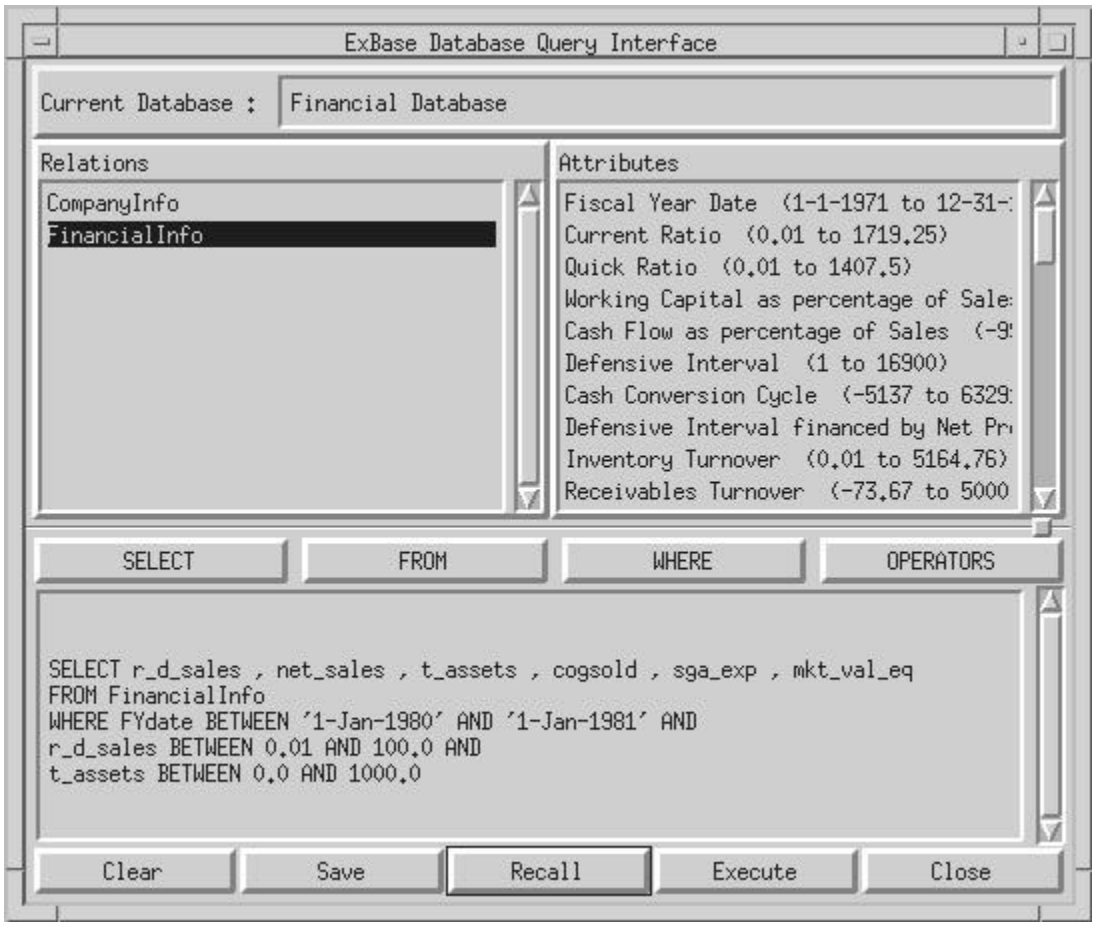

Figure 3-7. The Exbase Database Query user interface.

Figure 3-8 shows the Visualization Manager user interface that contains lists of selectable local database views and visualization views, along with their associated metadata. The analyst may choose an existing visualization view to redisplay, or a local database view and a visual representation to receive a default data-to-visual primitive mapping and graphical display settings. Local database views are labeled sequentially. Visualization views are labeled with an identifier that designates the type of operation that created the view (e.g., **DS** for a display setting change or **VQ** for a visual query).

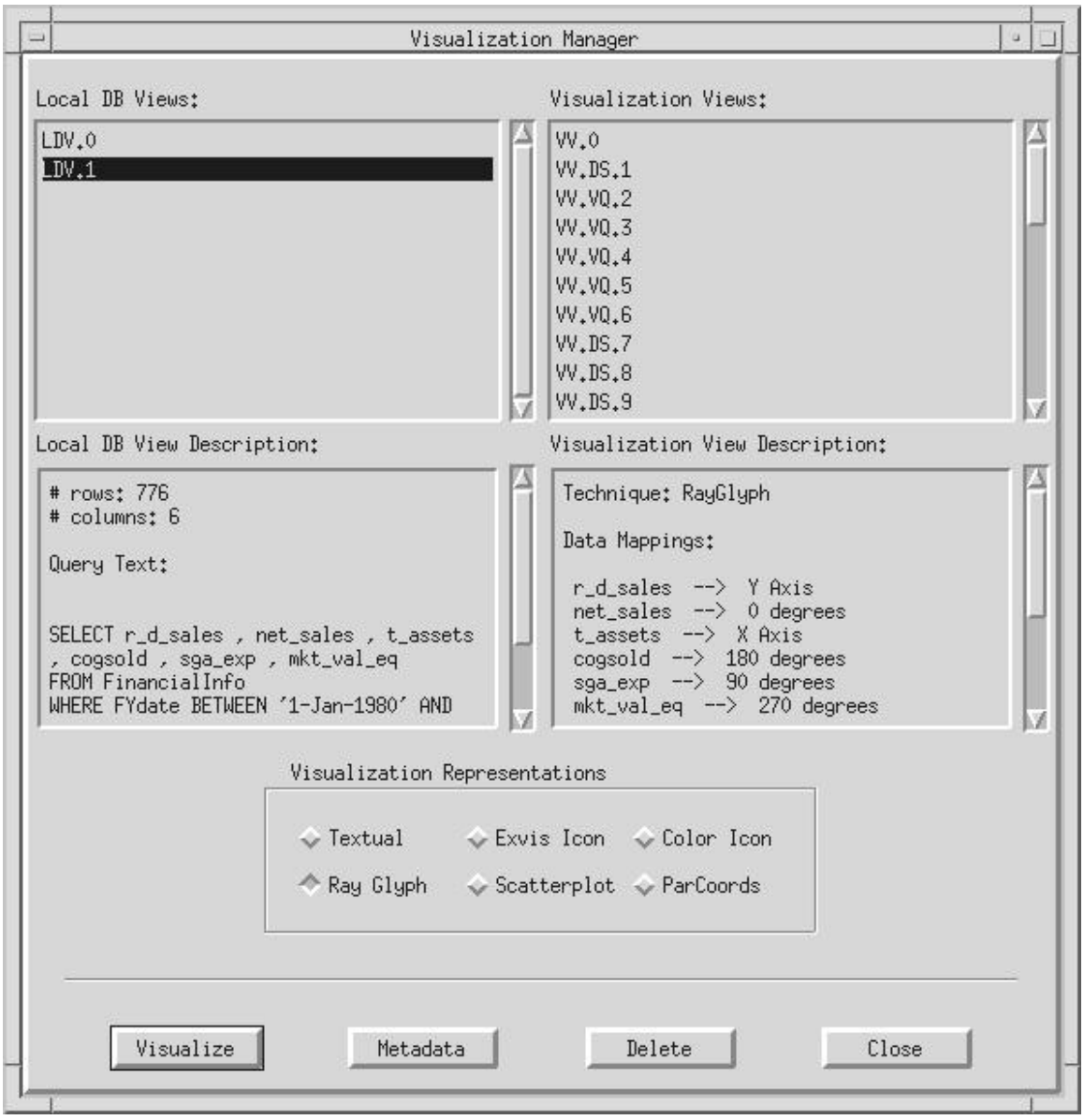

Figure 3-8. The Exbase Visualization Manager user interface.

Figure 3-9 shows the Visualization View user interface, and a visualization of the query result. 776 tuples were retrieved and the visualization representation is a rayglyph. Two attributes are mapped to (x,y), and the remaining to radial lines that emanate from the icon's center point. An enclosing polyline completes the rendering. The interface

contains data-independent (display) controls and data-dependent (data) controls. The data controls are range sliders showing a distribution histogram for each attribute, over a percentage of its range.

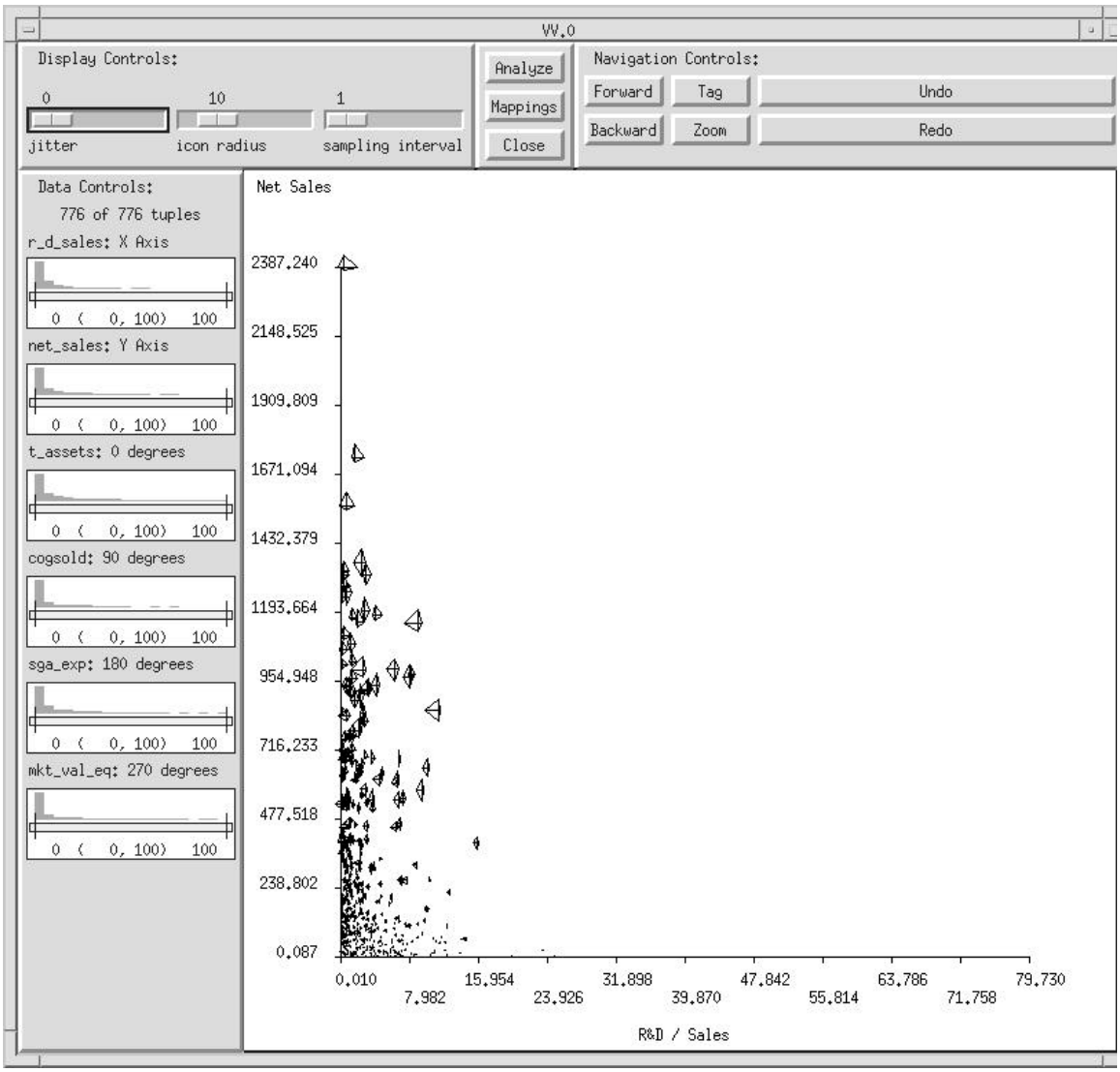

Figure 3-9. The Exbase Visualization View user interface, with query visualization.

Figure 3-9 shows an apparent correlation among the attributes, as evident from both the histograms and the visualization (icons become larger as *net sales* increase). Figure 3-10 shows a follow-up *zooming* interaction to make the visualization more clear.

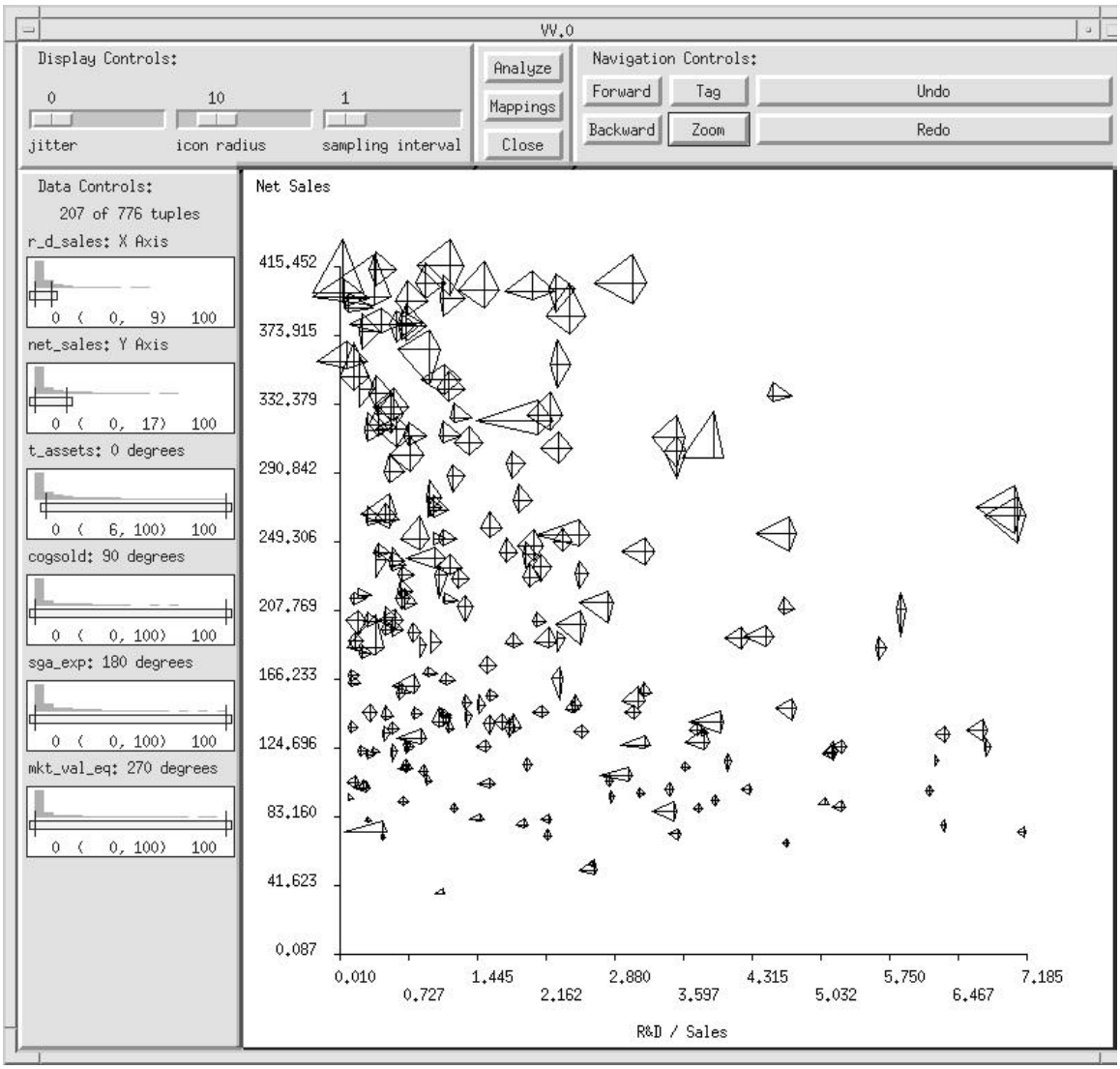

Figure 3-10. Zooming in on coordinate axes-mapped attributes and one other attribute.

In Figure 3-10, a visual zoom was performed on the two attributes mapped to the coordinate axes, *r_d_sales* and *net_sales*, and a data zoom was performed on *t_assets*.

The icons now fill the display, except at the bottom, where icons were removed whose *t_assets* value fell below 6% of its entire range. Figure 3-11 shows range query on each attribute, to approximately the lowest 10% of their ranges. This query eliminated 316 of the original 776 tuples visualized. The correlation is still apparent. All of the original outliers that may have skewed the visualization have been removed as the query was refined visually.

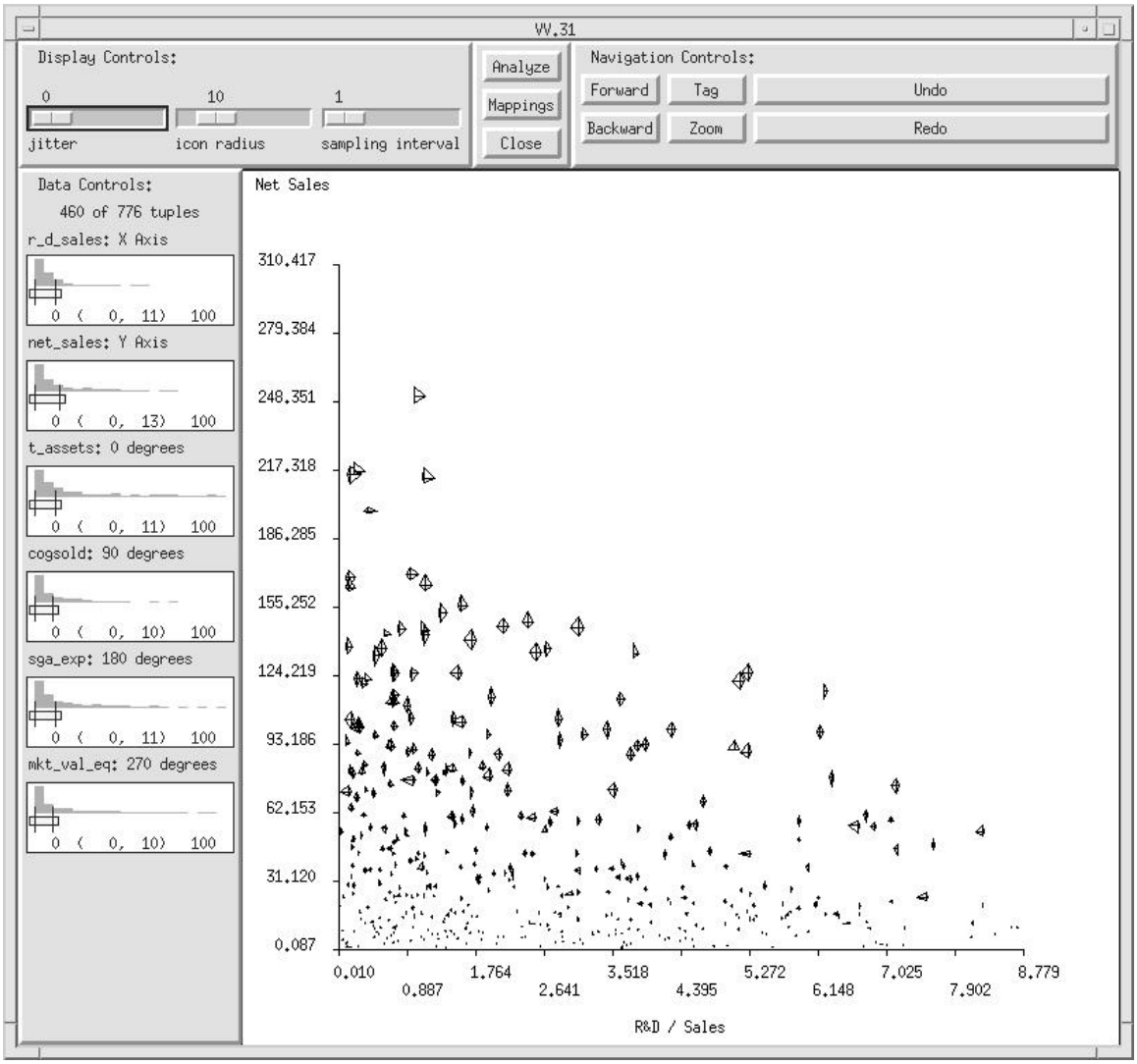

Figure 3-11. Zooming in on all data attributes.

Figure 3-12 shows the same local database view, but with a new data projection specified visually by a mapping user interface. This gives another perspective of the local database view, as different attributes are mapped to the coordinate axes. The correlation is still apparent, and there is now a small cluster centered around the point (71.0, 0.1).

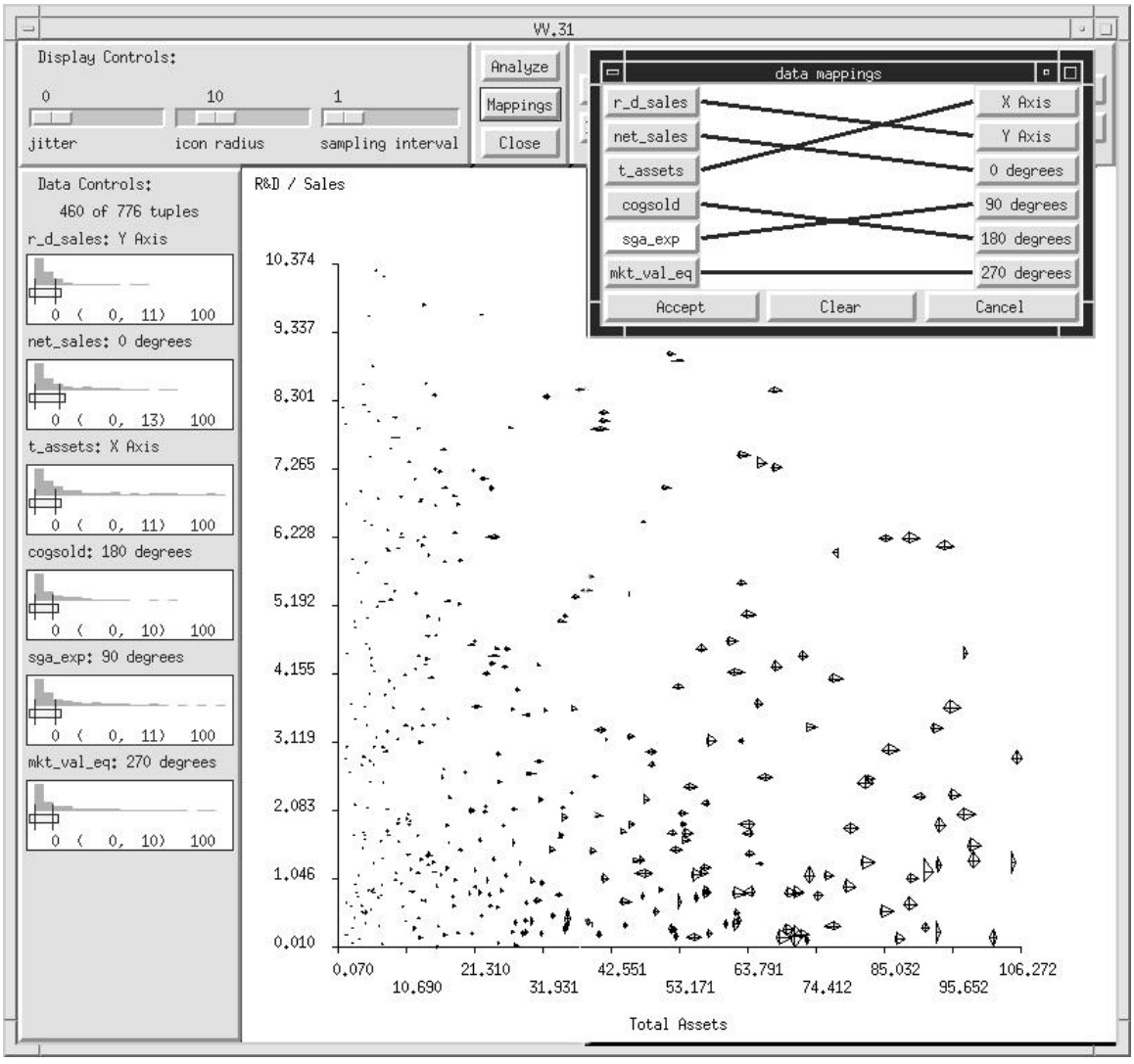

Figure 3-12. Changing the data-to-visual primitive mapping.

Figure 3-13 shows the effect of a follow-up data-independent operation, increasing the icon radius from 10 to 19. This interaction unveils a less prominent secondary cluster around data point (28.0, 0.1). This visualization also reveals that several of the icons may have missing or zero data values for some attributes (missing rays).

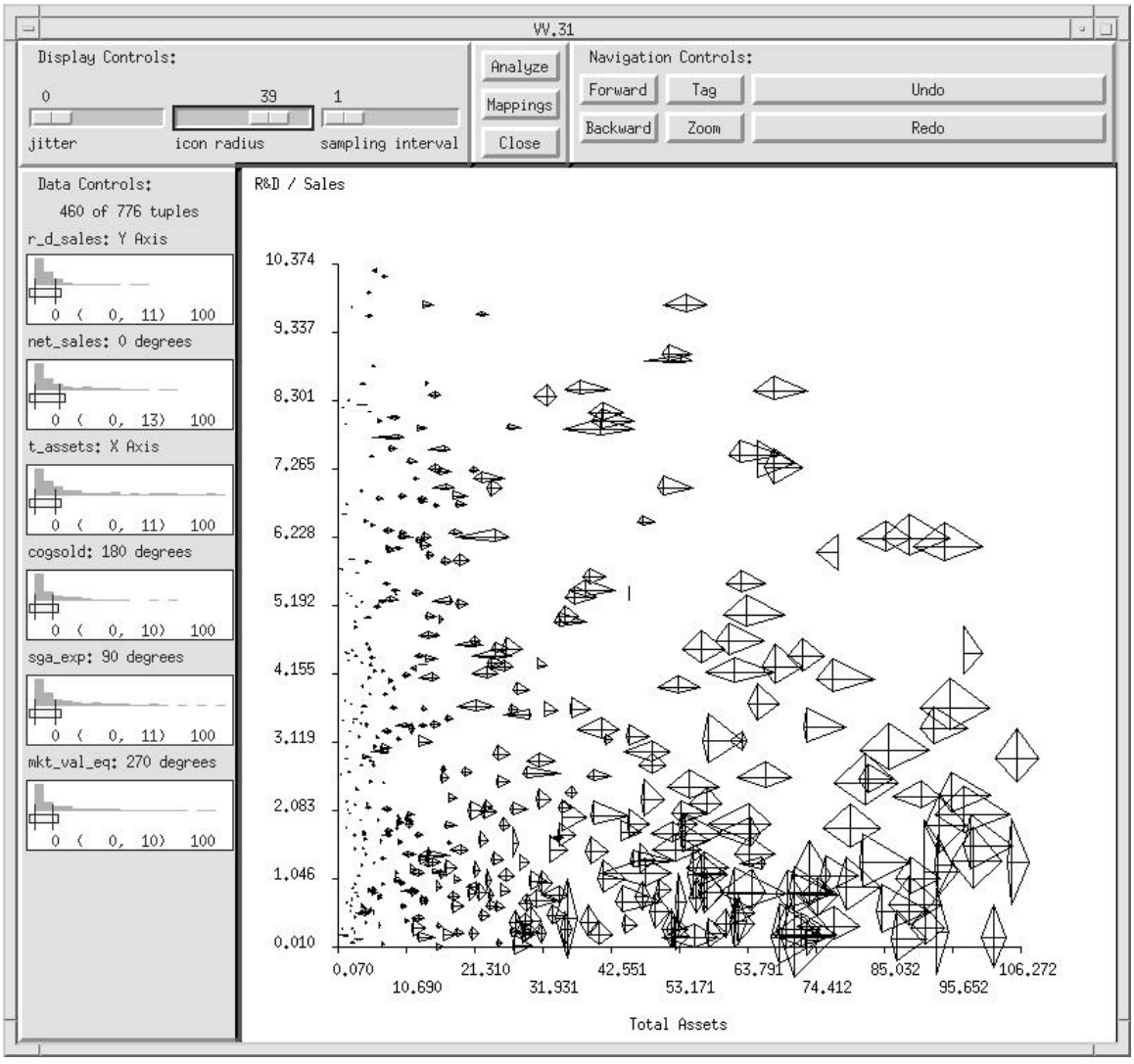

Figure 3-13. Increasing the *icon radius* graphical display setting.

Of course, only a domain analyst would be able to make sense of this database and the visualizations produced. The point here is that structures such as clusters and correlations can become apparent using the tools Exbase supports. Exbase maintains a session log of all user-data interactions, shown in Figures 3-14(a) and (b), primarily to supply data and metadata for the Visualization Manager user interface.

Figure 3-14 (a). The Exbase Session Log.

Figure 3-14(b). The Exbase Session Log (continued).

Figure 3-14(a) shows the start of a short Exbase session. Each operation by the analyst and system response is timestamped with a unique integer. Two separate queries are issued. Metadata describing database queries, local database views (LDBViews) and visualization views (VViews) are included. Starting at line 69, a data-dependent interaction is traced, a range query on *r_d_sales* and *net_sales*. Since each data range is normalized within the local database view, the range bounds are expressed as percentages of the entire range. Thus [0, 100] is full range, [0, 50] is the lower half of the range, and [50, 100] is the upper half of the range.

In the continuation of the session in Figure 3-14(b), a long sequence of data-dependent interactions with *sga_exp* predominates. This immediately indicates the attribute might be important to the analyst. Finally, a new data mapping is recorded before the session terminates.

3.4 Systems Model Summary

In this chapter, we have applied a user-centered analysis of database exploration to generate requirements for database exploration systems, and have described Exbase, a software architecture that addresses these requirements. We have shown how Exbase supports the integrated querying and visualization of relational databases by mapping the relational data model to the visualization system and supporting database operations at the visualization display. This integration is realized by the view concept, which is already defined in each component system.

Though the concept of a view is similar in both systems (a representation of the database), the nature of each system precludes their implementations from being similar. DBMSs are data-centric, emphasizing data selection, and visualization systems are graphics-centric, emphasizing graphics production. This influences the usefulness of each system for data exploration tasks. By combining both systems for data exploration, the appropriate qualities of each can be exploited. Exbase's layered approach to the integration localizes some interaction processing. Visualization display setting interactions only need to re-render data. Visualization mapping configurations (data projections) require a data mapping and rendering. Data selections require computation of a new database view at the local database view or database, a data mapping and a new rendering. Because we have an underlying database system, we can persistently store, manage and retrieve this session metadata that is generated during an exploration, for future analysis and exploitation, as described in [Lee95].

This integration is not without its costs, however. First, the translation between data models requires extra storage and processing for intermediate data representations. This decoupling of visualization from the persistent data impacts overall efficiency, but is necessary for the integration. During visual database exploration, there is a definite tradeoff between interactive response and the need for exploration services. The data selection service is computationally expensive, especially when external media need to be searched. Having some data selection capability in local memory mitigates problem. Exbase is currently bound to an underlying relational data model, and though our techniques can generalize to other data models having explicit connectivity and topology, their implementation would require much effort in translation and query processing. The focus here on relational data fills a great void in visualization by providing the means to visually explore large databases that are ignored by the visualization community at large.

Exbase currently supplies a single type of local database view, which is modeled on a relational table. This local view is used by a number of different visualization types. To manage all of these views, a visualization manager object is present to provide a listing of all database views and their sub-views (restrictions), and a listing of all the associated visualization views and their sub-views. Each view produces metadata that describes the view contents. A database view shows the query, the attributes and their ranges, and the size of the result. A visualization view shows the representation type, the display settings and the data mappings. It is important to note that the visualization view might not be an exact representation of the database. There are many places during data extraction, transformation and visualization where data can be altered, deleted or created. This is why it is important to retain the data lineage in a sequence of transformations and the query. It is currently an open question as to how fast and large the transform description metadata will grow. This will depend on the application and the user. Testing is required and efficient compression and pruning schemes need to be developed.

The systems integration model described in this chapter describes an architecture that allows the communication of data and operations between database and visualization systems. Its realization in the Exbase prototype, while serving an important validation purpose, has several limitations. The local database view is essentially a cache object, and many are created during a database exploration session. Only entire views are cached, there is no cache limit other than the size of virtual memory, and the replacement policy is supplied by the underlying virtual memory manager. The user must be aware of what is cached, as there are no database-resident views.

These limitations can be addressed with the creation of a client-side view cache, having a view replacement policy, and the ability to store view descriptions in the

database. Section 7.4.1 describes how we may approach these problems with information gathered from the second half of this thesis.

3.5 Towards a Data Exploration Process Model

This chapter describes a systems integration model based on primary database exploration tasks, and a realization of the model in the Exbase prototype. In addition, Exbase uses a session log that records the sequence of user-data interactions and supplies metadata used by the Visualization Manager to present the session to the data analyst. The session log can also be stored in the database and analyzed, to gain insight into the database exploration process.

The session log has limitations, however. The Visualization Manager user interface allows the analyst to select a previously created view, to explore further. This says something very important about the database exploration process - that backtracking is employed. The database exploration process is not only linear, but also hierarchical, and the hierarchy is somehow based on the data content of the views created by the analyst. A chief limitation of the linear session log is that it is difficult to ascertain a data context within the session; that analyst has no idea of the data relationships among the temporal sequence of views.

The remainder of this thesis is therefore devoted to the definition and evaluation of a database exploration process model.