CHAPTER 6

APPLYING THE GDE MODEL AND METRICS

So far, we have:

- created a database-visualization systems integration model and implementation based on a database exploration *task repertoire* (Chapter 3)
- defined a generalized data exploration (GDE) model to represent data exploration sessions (Chapter 4)
- developed numerous metrics that describe data exploration sessions in a data exploration domain-independent fashion (Chapter 5).

In this chapter, we tie the GDE model and metrics back to the implementation (Exbase) in order to show how the model and metrics are used to describe data exploration sessions. A *concrete* data exploration model must be realized to account for this influence. In the case of Exbase, this concrete model is a relational database, iconographic visualization model.

We first describe a general approach towards specializing the GDE model for a new *data exploration domain* (i.e., the data structures and task repertoire) and discuss how the metrics are applied, in general, with an example. We then proceed to specialize the GDE model for the relational database and iconographic visualization exploration domains, and show their canonical patterns in the graph. We conclude this chapter by using the specialized model and metrics to draw some conclusions about the kinds of graphs that data exploration sessions may produce in general, those that could be made by Exbase and an alternative exploration methodology, the *dynamic query* paradigm.

6.1 The General Approach

In order to analyze the data exploration process, the data structures and task repertoire of the concrete data exploration domain must be mapped onto the GDE model. This specialization defines the data exploration space such that distances can be measured. This section describes a general approach to be taken for any new data exploration domain, which involves a specialization of the GDE model, and guidelines for using the metrics to analyze the session with the newly-acquired data exploration capabilities.

6.1.1 Specializing the GDE Model

Specializing the data exploration domain consists of the following two steps:

1. define the data structures, and
2. determine the edge labeling scheme.

Defining data structures requires concretely defining data entities (Section 4.2) and data derivations (Section 4.3). For each data entity $e = (S_e, k, M_e)$, the data set S_e and the metadata M_e must be defined. This enables data direction to be determined and data distance to be measured. In addition to the examples described in Section 4.2, the data entity process metadata can also contain the vertex-based metric values from Section 5.4 and the analysis time t_A from Section 5.5.1.

For each data derivation $d = (S_d, t, M_d)$, S_d is already defined as a set of ordered pairs of data entities, so the acceptable data entity pairings and descriptive metadata M_d must be defined. Previous research has emphasized structural metadata for data entity-like objects over process metadata for data derivation-like objects. We take the opposite approach, since we are modeling a process, and emphasize process metadata.

Process metadata can be used to label forward derivation component graph edges, and assist in gaining insight into the complexities of the data exploration process. Among the available process metadata for labeling derivation edges, or visualizing them in general, are:

- derivation type identifier
- timestamp value
- derivation direction vector

The derivation type identifier is required, to show the type of derivation (e.g., query or visualization). The timestamp can be included where temporal information is required. The derivation direction vector is useful whenever the path within the data exploration space is required. Since the contents of an edge label is dependent upon the information required by the session analyses, we use whatever labeling scheme is appropriate for the information we are trying to convey about the session.

6.1.2 Applying the Metrics

The forward derivation component graph can be analyzed both algorithmically and visually. In this thesis we emphasize the visual analysis, to give an intuitive appreciation of the *gestalt* of the data exploration process. Additionally, since the data exploration process is modeled as a graph, it is inherently visual.

Graph design is an inherently complex and difficult task; only simple graphs (i.e., planar graphs, or non-dense graphs $G = (V,E)$ where $|E| < |V|^2$) can be easily understood. Further complicating matters is the fact that the forward derivation component graph can be configured such that it is a visual metric structure for the data exploration process (cf.

Section 5.2), which tends to consume more display space than other configurations. In this chapter, we attempt to show graphs that are comprehensible in both style (how they are displayed) and substance (the data exploration metric information they contain).

Analyzing data exploration comprises a static (i.e., structural) analysis of the forward derivation component graph using the metrics defined in Section 5.4, and a dynamic (i.e., temporal) analysis using the data exploration calculus defined in Section 5.5. Since the metrics are structural, they lend themselves to visual analyses. Applying the calculus, on the other hand, is better served with algorithmic support, because of the increased bookkeeping that maintaining timestamps requires, and because explicit state changes (which are not accounted for in the visualization of the forward derivation component graph) must also be maintained.

Figure 6-1 shows a hypothetical data exploration summary graph. All derivations are single-edged, which means that vertices having multiple incident derivations ($\mathbf{M}_{\text{result}} > 1$, such as vertices C, D and E) can be reached via *at least* $\mathbf{M}_{\text{result}}$ independent derivation paths from the database E_U, or some common ancestor vertex. There may be additional paths if any intermediate vertices also have $\mathbf{M}_{\text{result}} > 1$. This means that there should be some data overlap among those independent paths. In this example, the forward derivation component graph essentially shows $\mathbf{M}_{\text{source}}$ and $\mathbf{M}_{\text{result}}$ for each vertex in the session.

As the graph is constructed during the session, $\mathbf{M}_{\text{source}}$, $\mathbf{M}_{\text{result}}$, $\mathbf{M}_{\text{visit}}$, $\mathbf{C}_{\text{source}}$ and $\mathbf{C}_{\text{result}}$ can be automatically updated for each vertex. Note that only the vertices participating in each new derivation need updating of all of their vertex-based metrics. Thus, all that is required to compute these measures is to sweep through the forward derivation component graph once, an O(V) process where V is the total number of vertices. No additional space is required.

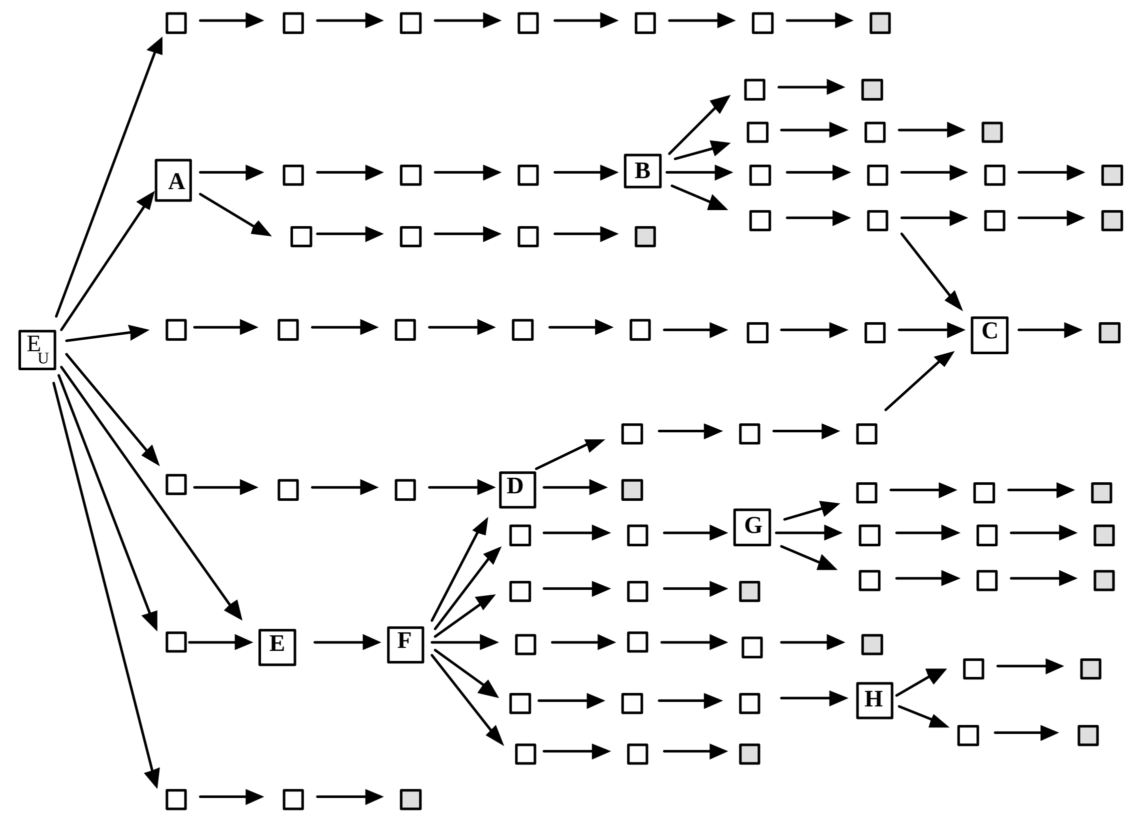


Figure 6-1. Data exploration session summary graph example.

We now proceed to identify the structures that are defined by the metrics.

Vertex-Based Metrics

Quasi-source vertices include E_U, and possibly vertices A, B, F, G and H. The threshold criteria needs to be established for identifying source vertices, based on Table 5-2. In comparing the possible quasi-source vertices, aside from E_U, F is clearly a quasi-source vertex because it has the highest number of exiting edges. B might be a quasi-source vertex as well, by similar reasoning. *Terminal* vertices (17 of them) are shaded. Vertex C may be a *quasi-terminal* vertex, though there may be others resulting from explicit state changes. There are numerous *sequential* vertices (those with exactly one incident and one exiting edge), and a single 2-sequential vertex (D). In general, it may be difficult to determine *k-sequential* vertices because cycles and repeated sequences are not shown directly in the forward derivation component graph; timestamps are required to insure there are no cycles present.

From the visualization it is difficult to determine landmark vertices unequivocally, since $\mathbf{M}_{\text{visit}}$ is not displayed (though it is available). We can assume, however, that in the absence of having $\mathbf{M}_{\text{visit}}$ displayed, that $\mathbf{M}_{\text{source}}$ is a good indicator of landmark vertices. Possible landmark vertices include vertices A, B, F, G and H. Taking the total number of vertices in the graph into account, it seems more likely that vertex F is the sole landmark vertex, since it has many more exiting edges than the other candidates. Of course, E_U is a landmark vertex.

Vertices E and C are *convergent* vertices, and vertices A, B, F, G and H are *divergent* vertices. If we were to define a binary relation "more divergent" signified by the symbol $>_{div}$, we can write the following: F $>_{div}$ B $>_{div}$ G $>_{div}$ {A, H}. Note that in this example, there are similarities between source, landmark and divergent vertices.

Path-Based Metrics

Here, we take measurements of path-based metrics at several locations in the forward derivation component graph of Figure 6-1. Results are summarized in Table 6-1. For the entire session, we measure the following:

number of vertices = 83   number of edges = 84
number of linearly independent paths = 31
maximum path depth = 9
average path depth = 7+7+8+9+9+5+9+9+9+5+8+2+8+8+8+5+6+
8+8+5+9+3+9+9+9+6+7+9+9+6+3 = 222/31 = 7.161
average path breadth = 6+7+7+10+11+12+12+9+8 = 82/9 = 9.111
cyclomatic complexity = 85 - 83 + 2(20) = 42

For the forward derivation component path rooted at vertex **A**:

number of vertices = 24   number of edges = 23
number of linearly independent paths = 6
maximum path depth = 8
average path depth = 4+6+7+8+8+8 = 41/6 = 6.833
average path breadth = 2+2+2+2+4+4+4+3 = 23/8 = 2.875
cyclomatic complexity = 23 - 24 + 2(5) = 9

For the forward derivation component path rooted at vertex **B**:

number of vertices = 16   number of edges = 15
number of linearly independent paths = 5
maximum path depth = 4
average path depth = 2+3+4+4+4 = 17/5 = 3.4
average path breadth = 4+4+4+3 = 15/4 = 3.75
cyclomatic complexity = 15 - 16 + 2(4) = 7

For the forward derivation component path rooted at vertex **F**:

number of vertices = 38   number of edges = 37
number of linearly independent paths = 10
maximum path depth = 6
average path depth = 4+2+6+6+6+3+4+6+6+3 = 46/10 = 4.6
average path breadth = 6+7+6+6+5+5 = 35/6 = 5.833
cyclomatic complexity = 37 - 38 + 2(9) = 17

feature	**verts**	**edges**	**paths**	**max(D_{path})**	**avg(D_{path})**	**avg(B_{path})**	**C_{path}**
G_{fwd}	83	85	31	9	7.161	9.111	42
A	24	23	6	8	6.833	2.875	9
B	16	15	5	4	3.4	3.75	7
F	38	37	10	6	4.6	5.833	17

Table 6-1. Metric summary for the graph of Figure 6-1.

Having numeric descriptions of the paths and graph allow comparisons. Using the Postulates defined in Chapter 5, we can say that the path rooted at vertex F is more exploratory than all the others because it has the greatest average path breadth. Similarly, the path rooted at vertex A is more focusing because it has the smallest weighted path breadth (it also has the greatest maximum path depth).

The cyclomatic complexity for each forward derivation component path selected seems to be related to the number of linearly independent paths and path breadth. This is not unexpected. Overall data exploration complexity is based heavily on explicit state changes, and the many branches that can be taken during an exploration session. What significantly influences the cyclomatic complexity is the number of explicit state changes from the forward derivation component graph, which requires calculating $\mathbf{M}_{\text{visit}} - \mathbf{M}_{\text{source}}$ for each vertex. This can be performed during graph construction, so the cyclomatic complexity, taking explicit state changes into account, can be easily determined. One indicator of this enhanced notion of cyclomatic complexity can be the number of terminal vertices, which are known to have at least one explicit state change out of them. Adding the number of terminal vertices to the total edge count in the complexity metric will yield a variety of values based on graph structure.

As with the vertex-based metrics, computing the path-based metrics requires a single sweep through the forward derivation component graph, an O(V) operation where V is the total number of vertices.

The Data Exploration Calculus

Reasoning about a data exploration session by applying the data exploration calculus depends largely on the concrete data exploration domain. In general, analyzing an exploration session graph using the calculus involves determining regions where derivation rates and possibly data distances are changing rapidly. By traversing the forward derivation component graph in a breadth-first manner, using the lowest available timestamp as the traversal criteria, the graph will be traversed in temporal order, and derivation rates can be easily determined. Incorporating this information with the metrics from the graph visualization would be very helpful in analyzing the dynamic nature of the session. This is a visualization issue that we do not address here.

Detecting temporal clusters in graph structures signifies derivation acceleration. Temporally dispersed paths signify deceleration. Discontinuities (where the next derivation chosen is an explicit state change) indicates terminal and quasi-terminal vertices, and a change in the derivation direction. A limitation of the forward derivation component graph is that explicit state changes are not directly stored in derivation edges, so their temporal behavior is not readily available. However, just as with determining the number of explicit state changes at a vertex, this value can be computed from the graph once the next state is determined during the traversal. Alternatively, the c- or s-derivation graph representation can be used as the base representation, and the forward derivation component graph can be used for visualization and most analysis purposes.

Using the calculus, we would expect landmark vertices to exist in regions where derivation rates are slow, because the analyst would be taking the time to determine how to derive anew from the landmark vertex. Alternatively, we would expect sequential vertices to exist in regions where derivation rates are constant or increasing.

For example, consider the subgraph of Figure 6-1 reproduced in Figure 6-2, having the indicated temporal information.

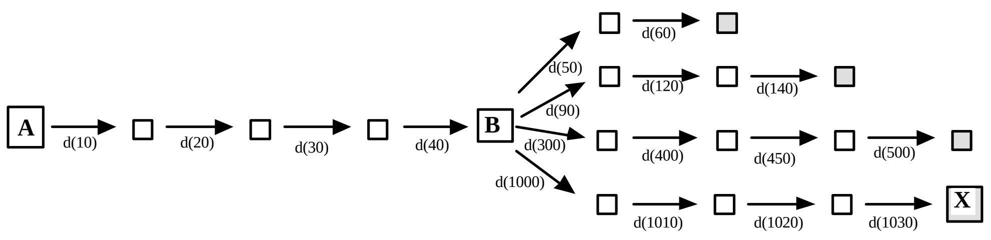


Figure 6-2. Applying the data exploration calculus.

In determining the temporal profile of the path <A, ..., B, ..., X>, the derivation rate is a constant 10 seconds per derivation except at vertex B, where there is a noticeable change that affects the average interaction speed of the path. Just considering that path in isolation immediately suggests that something important happened at vertex B: either the analyst took a very long time to analyze it, or wandered off to explore some other region of the data exploration space before returning to vertex B to continue the path to vertex X. The data exploration calculus is meant to pinpoint such regions within the exploration session graph.

Cycles in the Session

Cycles in the data exploration session arise from explicit state changes, and are often coupled with repeated derivation sequences. Cycles are significant because they indicate locations in the data exploration space where the analyst may be having difficulty, or needs a reminder of some observed phenomena. While cycles are stored explicitly in the data derivation graph representation (i.e., in backward derivations), they are not in the forward derivation component graph. As with determining temporal clusters, it is advisable to use the c- or s-graph representation in addition to the forward derivation component graph.

The forward derivation component graph does show an interesting pattern that is a variant of a cycle, where paths overlap at a single vertex - the landmark vertex. Consider the forward derivation component path rooted at vertex B in Figure 6-1, where each linearly independent path is related at vertex B. Depending on the derivations applied along each path, they could be very close together in their data content.

6.2 The Relational Database Specialization

In this section, we specialize the GDE model for relational databases by modeling relations, or local database views (Section 3.3.3) as data entities, and database queries as data derivations. We then devise an appropriate edge labeling scheme. Finally, we show two canonical relational database forward derivation component graph patterns and comment on their metrics.

6.2.1 Defining the Data Structures

Representing the local database view by a data entity structure is a straightforward task. The *extension* of the local database view is represented by the data set S_e, making S_e a relation that is composed of tuples containing single-valued attributes. The identification metadata consists of the name of the local database view, the names of the attributes contained within the relation, the timestamp and key value. The key value here is different than the relational database concept of a key, since the data entity key is non-unique. The actual database key value for a data entity would be (timestamp, key). Since a relational database query produces a single relation as output, the data entity key value for local database views is always 1 (k=1).

Local database view structural metadata includes the number of tuples and attributes, tuple size, and statistical attribute summaries. Statistical summaries, crucial for analysis, can be thought of as “analytic structures” that describe the structure of the data with respect to the data exploration space (Section 5.1). The process metadata contains a reference to the query that created the local database view (the query describes the local database view at a high level), the total analysis time t_A and the vertex-based metrics.

Representing the database query by the data derivation structure is also straightforward. The data set S_d is a set of ordered pairs of the form (database relation, local database view) or (local database view, local database view) that expresses the mapping from query input (a set of relations or local database views) to query output (a single local database view). Identification metadata includes the query text, the timestamp value and the data derivation key value. Since queries can map several relations onto a single output relation, $k \geq 1$. Structural metadata includes the size of the query text, the number of attributes and relations referenced, and perhaps a parse tree of the query. Process metadata includes the execution time, t_c in Section 5.5.1.

6.2.2 Determining Edge Labels

Figure 6-3 shows examples of an edge labeling scheme for a representative set of database queries, where the derivation type is identified by q, with a derivation component vector subscript, and a timestamp. The timestamp may be eliminated if we assume that t_c is sufficiently small, meaning the response time for the computer to compute the derivation is negligible when compared to t_A, the time spent by the analyst in analyzing the result.

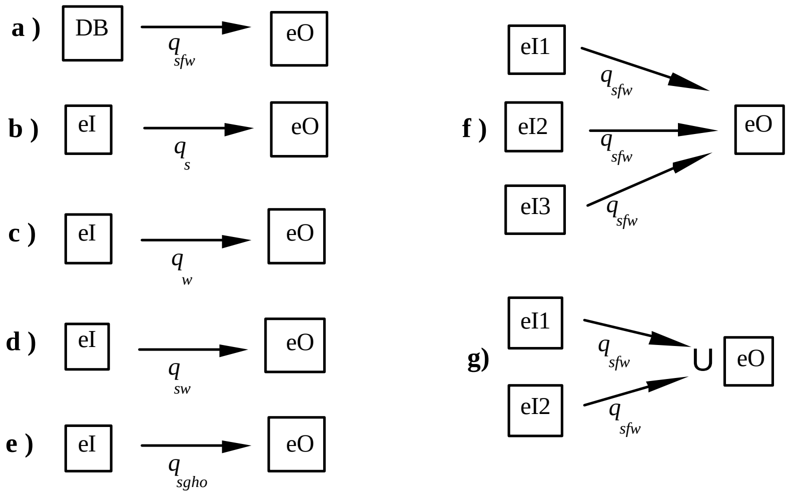


Figure 6-3. Relational database specialization labeling scheme.

Figure 6-3(a) shows how a SELECT-FROM-WHERE query over the database is labeled. The component vector *sfw* identifies the clauses used to select data from the database. Here, we still treat the database as a universal relation, though in reality it is a collection of relations. Figure 6-3(b) shows the labeling for a projection over a local database view. The subscript *s* signifies that only a SELECT clause was used to select attributes, while retaining all tuples. Figure 6-3(c) shows the labeling for a query that selects tuples from a local database view while preserving all attributes. The subscript *w* indicated that only a WHERE clause was used to select data. Figure 6-3(d) shows Figures 6-3(b) and 6-3(c) combined: an attribute selection and a tuple selection. The subscript *sw* indicates the use of SELECT and WHERE clauses only. Figure 6-4(e) shows the labeling for a query summarizes a database view with the HAVING and GROUP BY clauses, and also orders the result with an ORDER BY clause. The subscript *sgho* signifies that the SELECT, GROUP BY, HAVING and ORDER BY clauses contributed to the data selection.

Note that in Figures 6-3(b), (c), (d) and (e), the FROM clause was not used, since local database views contain a single relation; the FROM clause is implicit in these queries and targets the sole relation. Figures 6-3(f) and (g) show cases where the *f* subscript is required in the label. Figure 6-3(f) shows the labeling for a query over several distinct local database views. Figure 6-3(g) shows the labeling for the UNION of two database queries, using a union operator (∪) to signify the set operation at the resultant database view. Thus, when an *f* is present in the subscript of a database query over a local database view, there are other input local database views to the query.

6.2.3 Canonical Relational Database Patterns

There are two query sequences that are common to data exploration, data zooming and data panning. In this Section, we show the graphical patterns that these sequences create, and comment on their metrics. In this section, we assume t_c to be negligibly small, so it may be eliminated from edge labels to improve the readability of the graphs. In addition to the edge label identifying the query components used in the derivation, the *derivation directional unit vector* (Section 5.5.3) is used to indicate the data attributes specified in the query.

Data Zooming

Data zooming is the continued subsetting of a local database view by tuple selections. In the *zoom in* operation (Figure 6-4), the selected attributes of each query are kept constant, while the ranges on the attributes are successively narrowed. The primary query clause used is the WHERE clause.

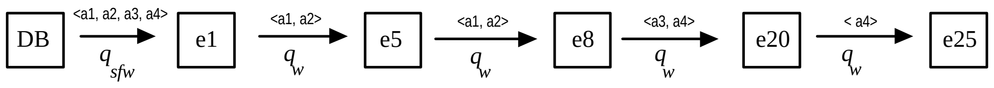


Figure 6-4. The *zoom in* interaction pattern.

A consequence of zooming in is that the size of each successive local database view is a proper subset of all predecessor local database views. In terms of the vertex-based metrics, zooming in creates sequential vertices. In terms of the path-based metrics, zooming in creates simple, linear paths having $\mathbf{B}_{\text{path}} = 1$ and $\mathbf{D}_{\text{path}}$ equal to the number of derivations in the path. Because long, linear paths are created, zooming in suggests a data refinement operation, where the analyst removes noisy data from the local database view via subsetting.

In the *zoom out* operation (Figure 6-5), the selected attributes of each query in the sequence are also kept constant, but the ranges on the attributes are successively expanded. As in zoom in, the WHERE clause predominates.

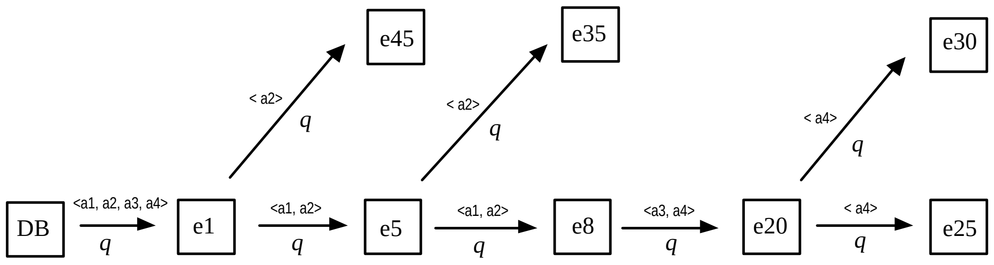


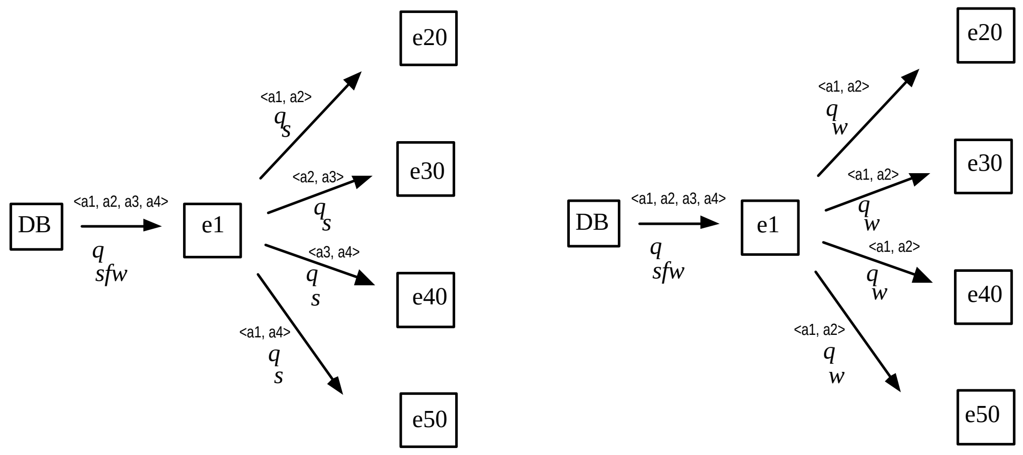
Figure 6-5. The *zoom out* relational database interaction pattern.

The zoom out operation creates a drastically different graph pattern than zoom in. In this case, the zoom out is actually a zoom in of some prior superset of the current data entity. A zoom out cannot be expressed over a local database view that does not contain the data to satisfy the query, or the result will be NULL. Thus, some predecessor local database view, or the database, must satisfy the query. In fact, since the relational data model is closed under relational operations, all zoom out branches could be moved such that they emanate from the database vertex. In Figure 6-5, the original zoom in is shown, and the three additional branching vertices correspond to the zoom out. First, the attribute a4's range is expanded, then attribute a2's range is expanded.

In terms of the vertex-based metrics, zoom out creates terminal vertices, such as e25 and e30, and source/divergent vertices, such as e1, e5 and e20. There are not enough branches on any vertex to distinguish a landmark vertex. In terms of the path-based metrics, zoom out increases the $\mathbf{B}_{\text{path}}$ value, while keeping $\mathbf{D}_{\text{path}}$ constant with the depth of the preceding zoom in. This implies a deviation from refinement, a backtracking to a larger data subspace, and therefore a more exploratory interaction. Both the metrics and the graph pick this pattern up quickly. Timestamps are critical to determining the exact path through the data exploration space.

Data Panning

Data panning is the continued subsetting of a local database view by attribute selections and *constant* range selections. These selections may or may not overlap in the data exploration subspace they define. Each successive local database view is a subset of at least one previous local database view, but not necessarily all previous views, as in the data zooming case. Figure 6-6 shows two examples of data panning.



a) panning on attribute **b) panning on tuple**

Figure 6-6. Relational database data panning interaction patterns.

In the attribute-based data panning example of Figure 6-6(a), a local database view is subsetted by specifying different vertical partitions of two attributes each, while keeping the tuple specification constant. The *s* subscript on the query labels signify that only SELECT statements were issued to create each new view. In the tuple-based data panning example of Figure 6-6(b), the attribute subset is kept constant, while the fixed data ranges of the attribute subset are changed.

In terms of the vertex-based metrics, data panning creates many terminal vertices, and a single source/divergent vertex that can be considered a landmark vertex, because there are many branching derivations out of the vertex, and many explicit state changes back to the vertex from the terminal vertices. In terms of the path-based metrics, $\mathbf{B}_{\text{path}} >> 1$, and $\mathbf{D}_{\text{path}} = 1$, which indicates more data coverage, and therefore an exploratory operation.

To summarize, in the relational database specialization of the GDE model, relations and local database views are mapped to data entities, and queries are mapped to data derivations. The primary derivation semantic is data subsetting, where derivation results are subsets of derivation inputs. Three primary interactions patterns were identified: *zoom in*, *zoom out* and *data panning*. *Zoom in* is a data refinement operation, *zoom out* slightly increases the exploratory nature of the operation and *data panning* is a high data coverage (i.e., exploratory) operation.

6.3 The Iconographic Visualization Specialization

In this section, specialize the GDE model for iconographic visualization. We model visualization views (Section 3.3.4) as data entities, and visualization transforms as data derivations. As with the relational database specialization, we analyze two common visualization interaction patterns and discuss their metrics.

6.3.1 Defining the Data Structures

The visualization view can be defined as the data entity $e = (S_e, k, M_e)$, where the database relation or local database view being visualized is the data set S_e. Thus, S_e in the data entity structure is a relation composed of tuples having single-valued attributes. Identification metadata consists of the name of the view, the names of the data attributes visualized, the type of visual representation scheme and the key value. In general, a visualization transform can produce more than one visualization view. Structural metadata consists of a reference to the local database view being visualized, and specifications on the image produced, typically a list of graphical primitives and the limits imposed on these primitives. Process metadata consists of the modeling transforms applied to the data set S_e, the total analysis time t_A and the vertex-based metrics.

Representing a visualization transform by a data derivation $d = (S_d, t, M_d)$ follows the earlier example of the database query. The data set S_d is a set of ordered pairs of the form (local database view, visualization view) to account for the traditional visualization transform, or (visualization view, visualization view) to account for incremental updates to a visualization. Identification metadata consists of the transform name, the visualization representation scheme (e.g., the leaf nodes of Figure 3-6), and the timestamp and key values. Structural metadata consists of the set of data-to-visual primitive mappings and the graphical display settings. Process metadata includes the execution time t_c.

6.3.2 Determining Edge Labels

In general, a visualization is specified by a representation scheme v_r, the data-to-visual primitive mappings v_m, and a set of graphical display settings v_G.

The *representation scheme* specifies, in the most basic terms, the type of visual representation for the data set (such as a particular line icon or color icon). It consists of an identifier and an n-tuple, *rep_name* $(vprim_1, vprim_2, ..., vprim_N)$, where each $vprim_i$ (i: $0 < i < N$) is a visual primitive that can encode a single data attribute.

The *data-to-visual mapping* specification consists of a set of ordered pairs *((att, vprim), (att, vprim), ..., (att, vprim))*, one for each attribute-visual primitive mapping.

The *graphical display settings* operate directly over the displayed data, and require a data-to-visual mapping and a representation scheme to already be specified. In general, there are three main categories: rendering environment settings, geometric environment settings and visual primitive settings. Rendering environment settings include color maps and lighting values. Geometric environment settings consist of values for the projection, window, viewport, scaling, field-of-view, etc. Visual primitive settings correspond to limits on their configuration, such as jitter ranges, maximum length or angular displacement for line icon segments.

Unlike the database query, which has optional components, each visualization transform must contain all three components (v_r, v_m, v_G). Incremental updates to v_m and v_G can occur as distinct derivations, though, and the existing values for the other components are used to complete the derivation transform's specification.

Figure 6-7 shows examples of an edge labeling scheme for iconographic visualization transform specifications, where the labels mimic the scheme for database queries. Note that we are using a new graphical representation to differentiate visualization views from local database views, an inverted triangle, and that there are many possible values for the graphical display setting subscript.

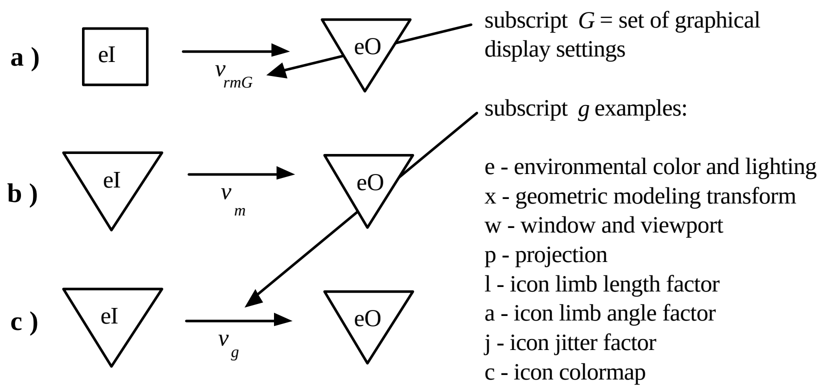

a)
eI
eO
v_{rmG}
subscript G = set of graphical display settings
subscript g examples:
e - environmental color and lighting
x - geometric modeling transform
w - window and viewport
p - projection
l - icon limb length factor
a - icon limb angle factor
j - icon jitter factor
c - icon colormap
b)
eI
eO
v_m
c)
eI
eO
v_g

Figure 6-7. Iconographic visualization specification labeling scheme.

Figure 6-7(a) shows a complete visualization transform that derives a visualization view from a local database view. The subscript *rmG* indicates that a new visual representation, data mapping, and complete set of available graphical display settings were specified to view the data. Figures 6-7(b) and (c) show derivations that transform one visualization view into another visualization view. In Figure 6-7(b), a new data-to-visual mapping is specified over an existing visualization view. Figure 6-7(c) shows a change in a single graphical display setting, as a visualization view is transformed into another visualization view.

6.3.3 Canonical Iconographic Visualization Patterns

There are two common visualization patterns that are similar to the database patterns of Section 6.2.3. Pan and zoom operations have well-known connotations in the spatial domain; it is very natural to manipulate a spatial data presentation by panning and zooming a window onto the scene. The general iconographic visualization display, discussed in Section 3.2.2, consists of a two-dimensional graphical display space, where data attributes are encoded onto the graphical primitives of the iconographic representation.

Spatial Zooming

Spatial zooming is very similar to data zooming, but the technique used to perform the zoom is different. In the case of spatial zooming, a geometric scaling transformation is applied to each visual primitive instead of an explicit data subselection and subsequent visualization. In the spatial *zoom in* operation of Figure 6-8, a repeated spatial scaling of the data creates proper subsets of a visualization view that is a containment relationship. The differences in the interaction pattern compared to Figure 6-4 are the visualization view replaces the local database view and the edge label includes the subscript *x* to denote a geometric transformation was applied. In this example, the spatial attributes x and y are the targets of the *zoom in*.

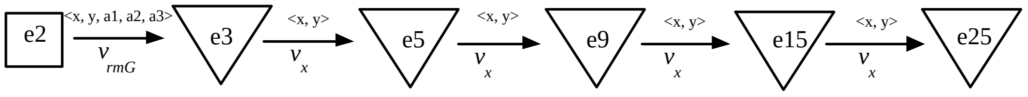


Figure 6-8. The spatial *zoom in* operation.

In terms of the vertex- and path-based metrics, the values are similar to those in the data *zoom in* example. Only the labels and data structures have changed. The spatial *zoom in* has the same significance as the data *zoom in*: a portion of the visualized data is refined by the removal of outliers. The distinction is in the method used to perform the zoom.

In the spatial *zoom out* operation of Figure 6-9, like its database counterpart, a slightly different pattern emerges. Following a *zoom in* operation, each successive *zoom out* is actually a *zoom in* of some predecessor visualization view that encompasses a larger spatial range. A stronger case can be made for placing all branching vertices of the zoom out directly off the local database view being visualized, since only a data-independent geometric scaling operation is applied. However, any derivation path to a targeted zoomed-out visualization view would then lose the zoom in operation that preceded the target view's creation. While this is a temporal issue, it is also a derivational issue, and we wish to retain as much of the derivation path as possible.

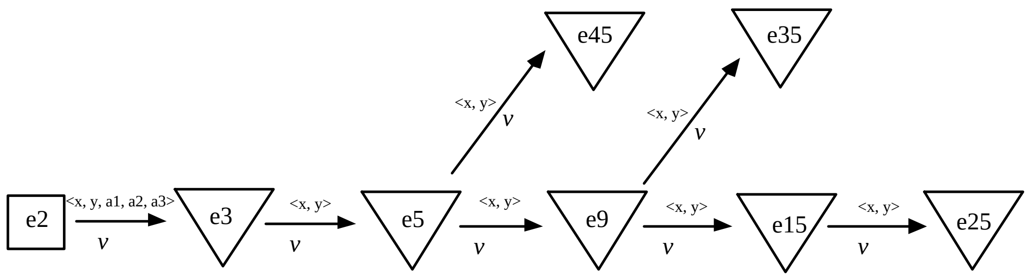


Figure 6-9. Spatial *zoom out* operation.

As with its database counterpart, the spatial zoom out indicates more spatial coverage (a larger spatial display) and therefore a more exploratory as opposed to refinement interaction pattern. The $\mathbf{B}_{\text{path}}$ increases while the $\mathbf{D}_{\text{path}}$ remains constant.

Spatial Panning

Spatial panning is very similar to data panning, but the data projection remains constant. The analyst retrieves proper subsets of a visualization view that *may* overlap each other. They do not have to overlap, but they are expected to, in order to preserve the sense of continuity between successive visualization views. Figure 6-10 shows two examples of spatial panning.

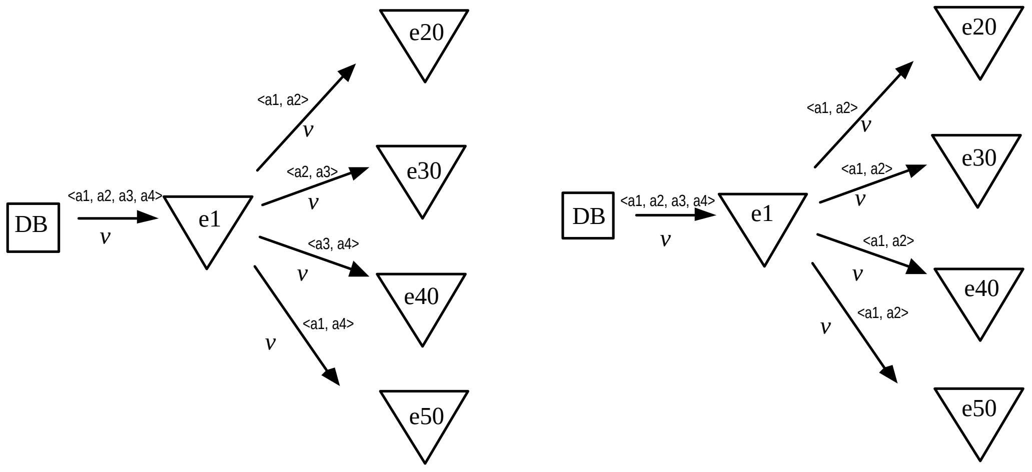


Figure 6-10. Spatial panning operation.

In the attribute-based spatial panning example of Figure 6-10(a), a vertical partitioning of the visualization view (i.e., attributes are selected) occurs while all the tuples remain constant. This is accomplished by specifying new data-to-visual primitive mappings, and in this case the patterns show the *grand tour* technique (Asimov 1985), where a sequence of visual data projections are continually and automatically presented to the analyst. The attribute panning can be overlapping, however, as the different permutations of possible mappings are sequentially created. This can be potentially more significant than database panning, because a certain data-to-visual primitive mapping permutation may be more effective at portraying data interrelationships than other permutations.

In the tuple-based spatial panning example of Figure 6-10(b), the data projection and the data-to-visual primitive mappings remain constant while a spatial translation geometric transform is continually applied. This can be thought of as moving a fixed window over a larger image. As in the database domain, many terminal vertices are created during panning, and a single source/divergent vertex may be a landmark vertex. For the path-based metric, these are exploratory patterns because $\mathbf{B}_{\text{path}} >> 1$ and $\mathbf{D}_{\text{path}} = 1$.

In the iconographic visualization domain, visualization views are mapped to data entities and visualization transforms are mapped to data derivations. As with the database domain. three primary patterns are defined; *spatial zoom in*, *spatial zoom out* and *spatial panning*. The spatial *zoom in* is a data refinement operation, while the spatial *zoom out* introduces more data coverage. In either case, the data projection (i.e., the data-to-visual mappings) are constant, and the zoom is relative to the attributes that are mapped to the coordinate axes of the display. The spatial data panning operation is a high coverage operation, as in the database domain.

6.4 Interaction Patterns in Database Exploration

In this section, we utilize both GDE model specializations in describing database exploration using relational database and iconographic visualization components. We also describe Exbase exploration, which requires a new type of derivation edge, an the dynamic query technique.

6.4.1 Canonical Database Exploration

The simplest database exploration is a modified visualization pipeline (Figure 2.7) that has provisions for both relational database and iconographic visualization GDE model specializations. Figure 6-11 shows this process, where *simulation data* is replaced by a relational database, *data enrichment and enhancement* is replaced by the SQL query, *derived data* is replaced by the local database view, *visualization mapping* is replaced by the visualization transformation, and the *abstract visualization object* is replaced by the visualization view. The rendering transform and resultant image are not shown explicitly, as they are components of the visualization view (Figure 3.6).

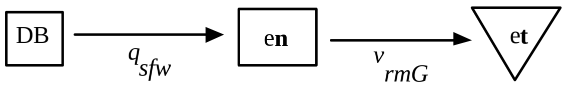


Figure 6-11. The fundamental database-visualization data exploration.

The traditional visualization pipeline of Figure 2-7 narrowly scopes visualization as a single task that is an output transformation, creating an image from a data set. This view of “visualization in the small” focuses on the visualization algorithm, and is an active research area. The input capabilities of a visualization are significant, however, as we have demonstrated with Exbase and modeled with a GDE specialization. This view of “visualization in the large” places visualization into a larger context, necessitating the additional input operations encapsulated in the visualization view object. Augmenting visualization in this manner elevates visualization to the same level as the database view because it can now be used as an input or output, yielding a type of *interaction closure*.

We can now use these process-level enhancements to the visualization pipeline to model database exploration sessions. Using Postulates 5.1 and 5.2, the derivation depth and breadth semantics, respectively, along with the GDE model specializations of Section

6.2 and 6.3, we can describe the kind of graph produced in a "typical" exploration session. This graph can then be used as a benchmark to compare actual exploration sessions.

When confronted with a new database to explore, the analyst discovers knowledge by determining the relationships among the data, within some framework based on the goals of the exploration. Typically, a user starts with an overview or summary view of the data followed by more detailed views. Thus, the analyst must initially cover as much of the data exploration space as possible before attempting to refine any particular data subset. The analyst might retrieve entire database tables, and then regularly sample every tenth tuple, or view distinct partitions of the database, for example. After the analyst has an idea of the gross structures in the database as a result of data coverage interactions, he can proceed to formulate precise derivations to extract and refine the embedded knowledge.

Postulate 5.1 states that linear structures in the forward derivation component graph (those having high $\mathbf{D}_{\text{path}}$ and $\mathbf{R}_{\text{path}}$ values) indicate data refinement. This corresponds to *zoom in* operations occurring at greater derivation depths. Postulate 5.2 states that branching graph structures (those having high $\mathbf{B}_{\text{path}}$ and low $\mathbf{R}_{\text{path}}$ values) cover large areas of the data exploration space, and indicate data exploration. This corresponds to panning operations occurring more frequently at smaller derivation depths, as shown in the summary graph of Figure 6-12.

The session graph of Figure 6-12 possesses more branching (i.e., panning operations) at smaller depths, and less branching (i.e., zoom in operations) at larger depths. The *zoom out* pattern is also present. Generalizing the nature of the fundamental data exploration pattern of Figure 6-12, we also expect more database panning and zooming to occur before visualization panning and zooming, since data sets must be prepared before they are visualized. Consolidation of local database views may occur, as several are queried at once to produce a single result.

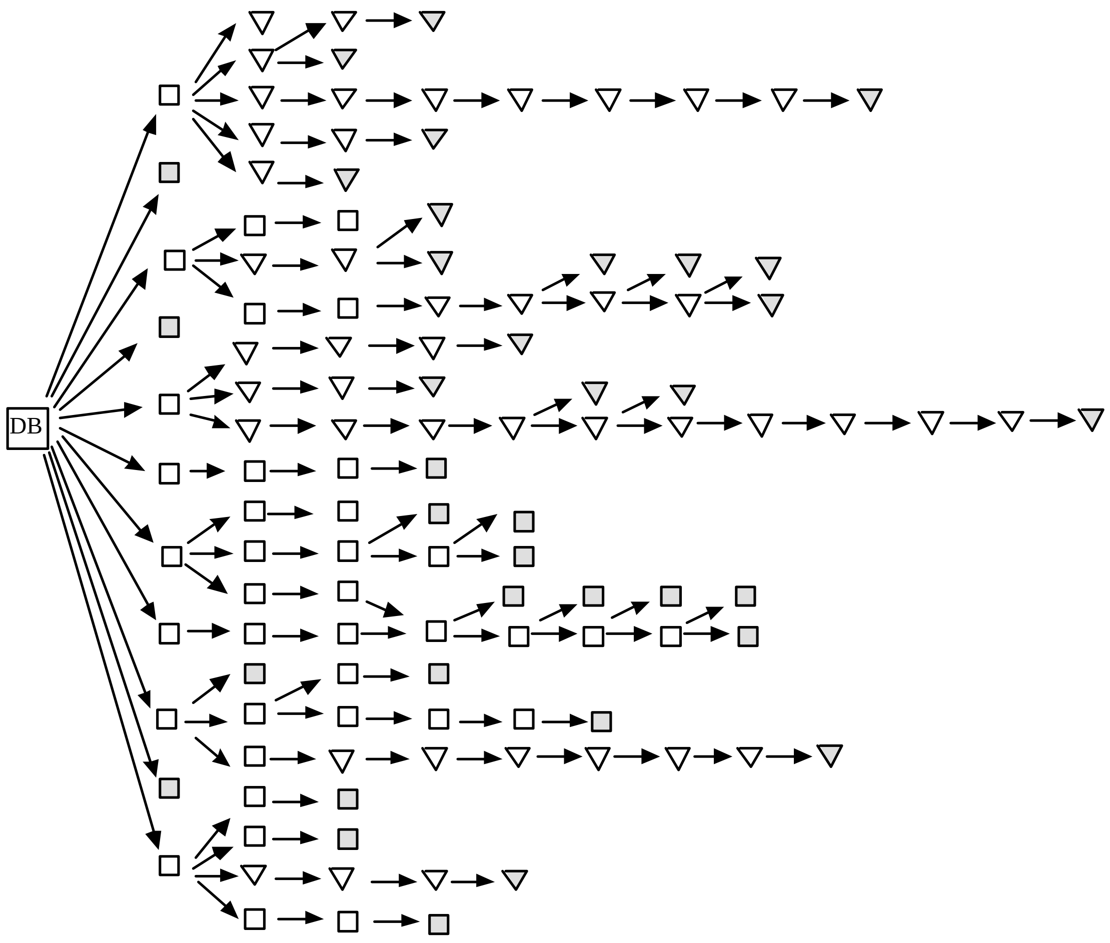


Figure 6-12. Canonical database exploration session graph example.

In some instances the analyst must perform some data refinement before more exploratory interactions can resume on a particular view. One example is a sequence of visualization display settings that are configured to create an image having enough compelling structure to provoke additional exploration. Another example is an image that is incrementally zoomed, towards an interesting subspace. The settings that are effective can then be saved, to be used again to explore the visualization view more quickly. Figure 6-13 shows a summary subgraph containing this type of pattern.

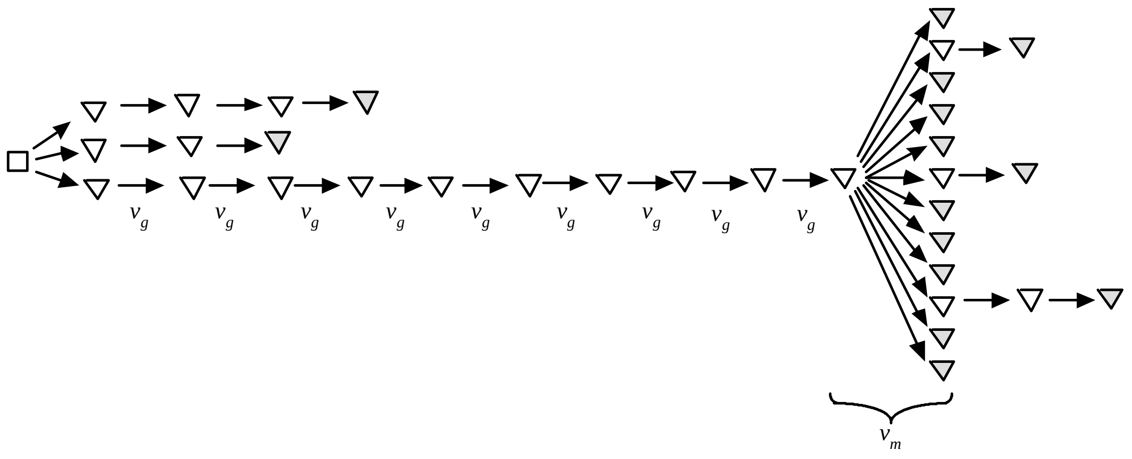


Figure 6-13. Refinement before exploration pattern.

The refinement before exploration pattern possesses a marked increase in derivation breadth at greater depths, but may do little in contributing to the overall average derivation breadth of the entire session. The reason for this possibility is that the interaction pattern from the source vertex of the panning interaction to its descendant, terminal vertices is likely to follow the canonical pattern of Figure 6-12, where breadth decreases at greater depths. Only a few of the results of the panning will be refined. It does, however, underscore the importance of the visualization component of the data exploration space. Exploration of the database involves exploration to find the presentation method and related settings that bring out the knowledge embedded in the data. Not only are there important data subsets, there are also important visualizations.

Both the database query and the visualization transform processes, analyzed in isolation, can be considered data refinement operations. A single query extracts a database subset based on a specific selection criteria, and a single visualization transform is intended to clarify the meaning of a large amount of numeric data. Analyzed in the larger context of a data exploration session, both processes can be classified as being either refinement or exploratory interaction patterns. Based on the data refinement and coverage semantic

Postulates developed in Chapter 5, we have identified two major types of database exploration session interaction patterns: exploration before refinement (Figure 6-12) and refinement before exploration (Figure 6-13). Based on common, accepted data analysis practices, the exploration before refinement pattern is dominant, while the refinement before exploration pattern is possible.

Once knowledge is discovered, the terminal vertices from which the knowledge was derived can be used to determine the set of simple forward derivation component paths (i.e., their data lineages) to the discovered knowledge. These paths can be classified as knowledge refinement paths, since they have high $\mathbf{D}_{\text{path}}$ and very low $\mathbf{B}_{\text{path}}$ values.

6.4.2 Exbase Exploration

The Exbase research prototype was designed to support a specific task repertoire: querying of the DBMS, visualization of query results, querying of query results and querying of visualizations. In relating the GDE model specializations with Exbase, the specializations do not account directly for the querying of visualizations, though Exbase does this by taking a data-centric approach towards supporting spatial pan and zoom operations.

Since pan and zoom operations are expressed in both database and visualization specializations of the GDE model, there are two methods to accomplish these operations: data-centric and graphics-centric. In the data-centric method, the data set underlying the visualization is subselected via a database query, and the resulting subset is then rendered. The database selection thus performs the trivial accept/reject phase of graphical clipping of those icons whose spatial axis-mapped attributes fall completely outside the viewing area. The icons sent to the rendering pipeline may have portions that overlap the clip boundaries, so the final rendering might need to be altered to make entire icons visible. In the graphics-centric method, modeling transformations (e.g., translation and scaling operations) are specified, and the graphics system performs all clipping. In the database specialization, pan and zoom are data-centric operations, in the iconographic specialization, pan and zoom are graphic-centric operations.

Exbase makes the distinction that each pan and zoom over a visualization creates a new SQL query that is stored as derivation metadata. This query could then be used again to recreate the local database view being visualized, if necessary. The sorted pointer arrays of Figure 3-5 are used to process range queries, and the projection vector of Figure 3-6 is used to process attribute selections in main memory. The graphical user interface of the visualization view forwards data requests to the local database view, which assembles a new query and performs the data selection. Thus, queries are now allowed over visualizations, requiring a new edge type to identify this important operation, shown in Figure 6-14.

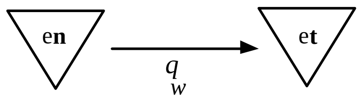


Fig. 6-14. Querying a visualization derivation edge.

Figure 6-14 indicates that a data selection was issued over a visualization view to create a new visualization view having the same visualization specification, but with a new, derived data set. There is an intermediate local database view implicitly created during the operation, but not shown, because the analyst sees this as a single operation from one visualization to another.

Exbase provides a limited set of queries over local database views: range queries over tuples. Attribute selection is performed at the visualization view. This set could be extended in the future, requiring additional SQL-parsing and main memory selection functions. While such capabilities would be useful, they may not be ultimately needed, since the data analyst might not need this functionality at the local database view level. User testing must be performed to determine the correct balance of DBMS-resident operations and main memory operations. At a minimum, however, range queries must be supported in memory because they are a fundamental exploration operation.

Exbase currently offers the following operations:

- textual queries submitted to the DBMS
- range queries over single local database views, through the visualization view GUI
- attribute projections, through the visualization view GUI
- visualization representation selection at the Visualization Manager GUI
- jitter, icon limb length scaling graphical display settings

Given these kinds of operations, most Exbase exploration sessions can be represented by the graph of Figure 6-15:

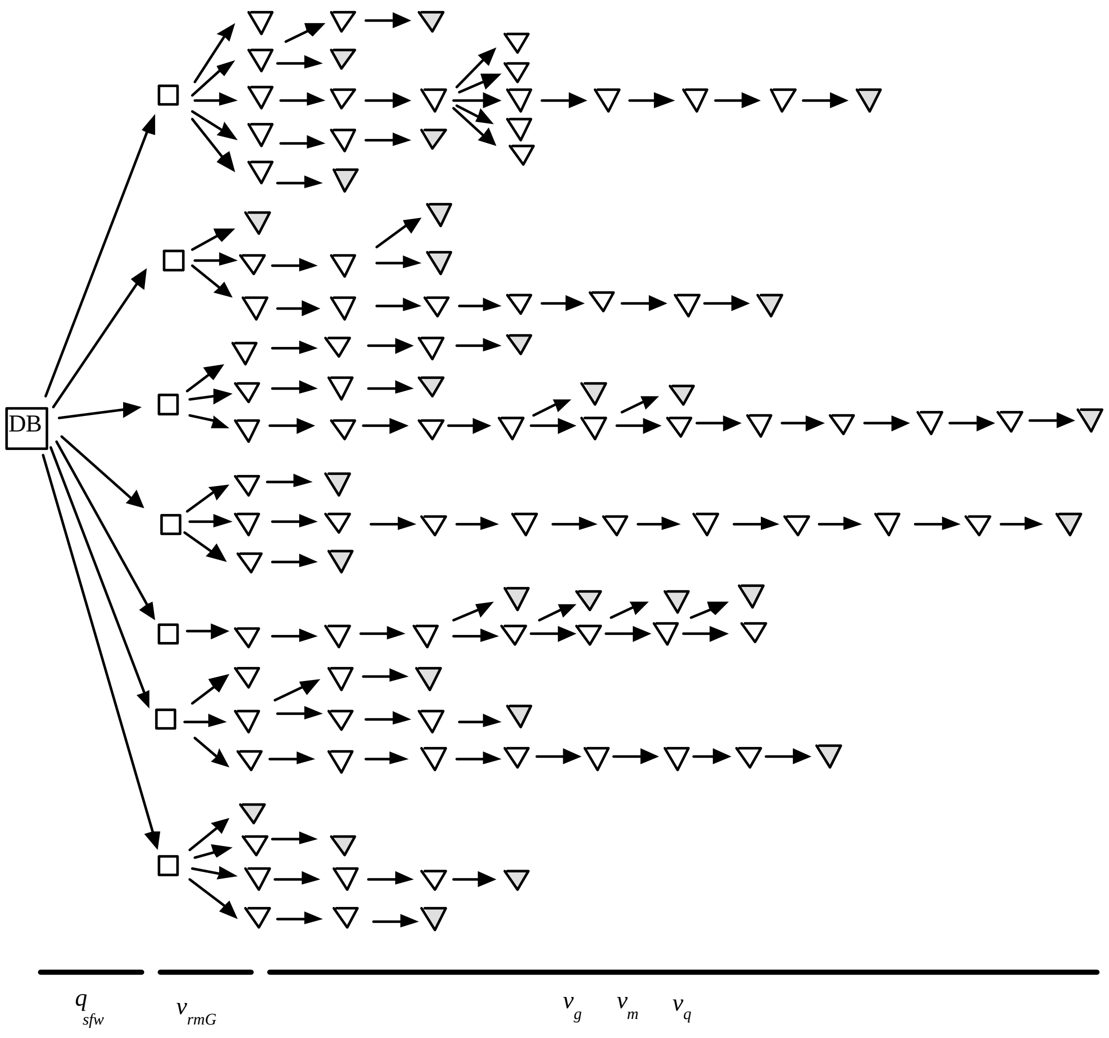


Figure 6-15. Canonical Exbase exploration session graph example.

The session graph of Figure 6-15 is actually a tree, because there are no derivations that merge local database views (such as a query over multiple local database views). Thus, the session graph is no longer a DAG. This is only due to the current limitations of Exbase. The graph still exhibits the same basic shape: greater breadth at lower depths. Note that only local database views are explicitly created at depth = 1, as SQL queries are issued to the database. All subsequent interactions are over visualization views. At depth = 2, the primary operation is v_{rmG}, and at greater depths operations are v_g,

v_m and v_q. Due to the operations offered by Exbase, there is a preponderance of pan and zoom operations over visualization views.

As currently implemented with X-Windows and Motif, Exbase's response to user interactions is highly dependent on the hardware platform, given the performance impacts of the windowing system. To realize acceptable performance under control widget manipulations, updates to the visualization (or session graph) do not occur until a control widget manipulation has completed (i.e., the mouse button has been released after the widget was altered). With respect to updating the visualization and session graphs, Exbase can be considered a "data push" technology, where data is sent from the interaction initiator when the interaction has completed.

6.4.3 Dynamic Query Exploration

Similar to Exbase exploration is the *dynamic query* technique (Ahlberg, Williamson and Schneiderman 1992), where discrete scatterplot visualizations are probed along multiple dimensions with range selection widgets. Exbase's data controls were actually inspired by this technique. The dynamic query update to the display is in real time with the data control widget manipulations, making this technique a "data pull" technology, where the data control stream is sampled at some interval to update the session graph.

In dynamic queries, the analyst does not have much control over graphical display settings. Only a single 2D scatterplot is typically supplied, leaving it up to the analyst's abilities at manipulating the data controls to determine any additional structure in the data by direct manipulation. The analyst has much greater control over the data that is displayed, with rapid response to any range query; special main memory techniques are implemented in this approach, and there is no DBMS to speak of.

Dynamic query session graphs, then, *should* contain many more vertices and edges in a given time period than Exbase session graphs. The dynamic query graph is dense in the number of vertices and edges that have to be packed into the visualization, due to the ease of interaction and rapid updating. They will have numerous landmark vertices with large numbers of descendent vertices, and numerous terminal vertices that are dead-ends. The main interaction patterns are zoom in and zoom out, as is shown in Figure 6-16. Panning on tuple only (the visualization representation is a 2D scatterplot) is also prevalent.

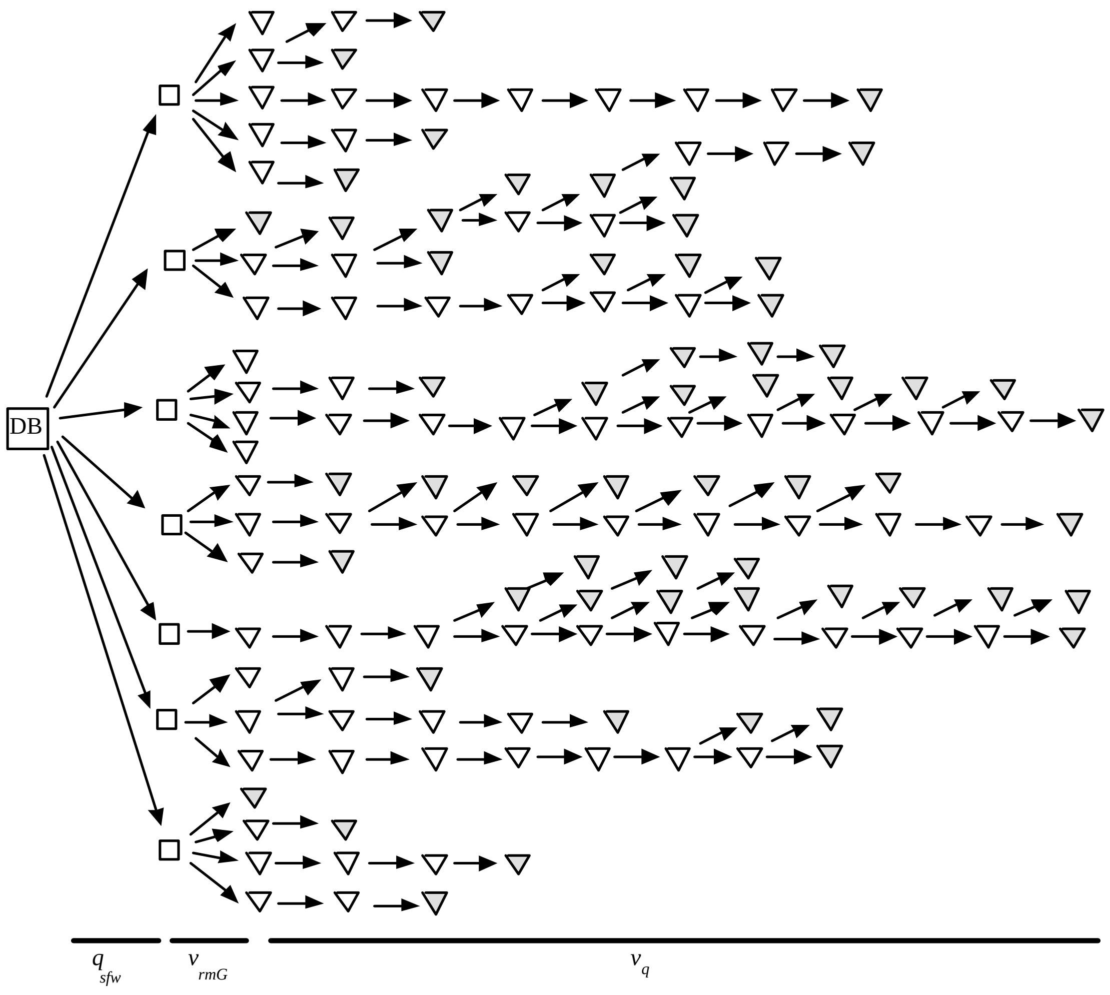


Figure 6-16. Dynamic query interaction pattern example.

Figure 6-16 shows what a dynamic query session graph might look like if a DBMS were available, since the technique is a main-memory technique that uses optimized data structures. In the Figure, the database is queried to produce local database views, the local database views are visualized as 2D scatterplots, and the visualization views are then probed via data selection queries. In the traditional dynamic query scenario, the graph only contains a single main path with branches off it, as the entire “database” (the flat file read into a main memory data structure) is visualized and probed via the data range sliders.

The average path breadth is high with this technique, because it is so easy to probe the database rapidly. This implies that the dynamic query technique is more exploratory than the Exbase technique, because there is more branching resulting from the effortless zoom in and out operations. In a similar fashion, data panning on tuples will create dense branching patterns that are highly localized. This makes sense intuitively when one considers that the analyst performs more interactions that cover the database component of the data exploration space. The visualization in this approach does little to harness the human perceptual system, as the Exbase approach does, so the search for structure within the database is highly dependent upon database probing.

6.4 Chapter Summary

In this chapter, we have applied the GDE model and its metrics towards describing database exploration in the relational database and iconographic visualization domains. This application involves mapping the concrete data structures and derivation structures of those domains to the abstract GDE structures and devising a simple edge labeling scheme.

Data exploration sessions are represented by forward derivation component graphs, and we use graphical representations described in Section 5.2 to depict data exploration sessions and their interaction patterns, based on the information we are trying to convey. Similarly, we devised a derivation edge labeling (visualization) scheme. We then integrated the GDE model, metrics and exploration domains to describe fundamental interaction patterns having defined graphical representations. These fundamental patterns correlate with the notions of *data refinement* and *data coverage*, and exhibit the semantic properties of Postulates 5.1 and 5.2.

We used the expanded model to represent database exploration using integrated relational database-iconographic visualization techniques. We described the canonical session graph, built on the Postulates and interaction patterns of the concrete domains, and an additional complementary pattern. The descriptions of these patterns are based primarily on the derivation path-based metrics which are built upon the lower level vertex-based metrics. We finally showed how the model can capture Exbase explorations and dynamic query explorations, and used the model to differentiate between the two techniques.