A Visual Approach to Multimedia Querying and Presentation*

Isabel F. Cruz
Department of Computer Science
Worcester Polytechnic Institute
ifc@cs.wpi.edu
www.cs.wpi.edu/People/faculty/ifc.html

Wendy T. Lucas
Database Visualization Research Group
Tufts University
wlucas@cs.tufts.edu
www.cs.tufts.edu/~wlucas

Abstract

Multimedia data has become readily available from a variety of resources, such as the Web, to users (ranging from naive to sophisticated) who need to select and to present the data in a way that is meaningful to their particular applications. Delaunay^{MM} is our framework for querying and presenting multimedia data stored in distributed data repositories, including the Web. It is unique in combining user-defined layouts with ad hoc querying capabilities, thereby enabling users to tailor, in a simple way, the layout of virtual documents composed of retrieved multimedia objects. In this paper, we focus on the object-oriented data models, on the declarative query languages, and on how the results of the queries to disparate resources are integrated to form coherent user-defined documents.

1 Introduction

Database management systems (DBMSs) have traditionally enabled the storage, retrieval, and presentation of tabular datasets primarily for professional applications. The recent phenomenon of the World Wide Web, the associated advent of multimedia data types (i.e., images, audio, video, as well as text), and the emergence of widely available distributed databases have significantly changed the ways in which we interact with data. In addition, a significant shift in who is doing that interacting and for what purposes has occurred; for the first time in the history of the computer, non-professional users have access to vast quantities of information via modem-equipped computers found in homes, schools, and offices.

To address these new requirements, concepts and tools are needed that enable users ranging from naive to sophisticated to not only select the information they need but also to present it in a way that is meaningful to their particular application [17, 21]. Our framework for querying and presenting multimedia data stored in distributed data repositories, including the Web, is called Delaunay^{MM}. Its uniqueness lies in its combination of user-defined virtual document layouts with the ability to define document content through ad hoc queries to multiple repositories.

Delaunay^{MM} is a multimedia extension to the Delaunay Database Visualization System [9], an interactive, constraint-based system for visualizing object-oriented databases. Delaunay users pictorially specify, in an intuitive yet formal way, the visualization of database objects. By arranging graphical geometric objects and graphical constraints, users form a "picture" that specifies how to visualize data objects belonging to a class.[1] Following a similar approach, users of Delaunay^{MM} visually represent the spatial layout of the data to be retrieved from distributed multimedia repositories.

The Delaunay^{MM} document layout model defines a virtual document as being a set of user-specified style sheets. Therefore, the layout of a document is based on one or more style sheets (e.g., for the layout of the title page or of the chapter pages). Within the document, a set of pages is associated with each style sheet, which serves as a template for the layout of these pages. The user associates queries with the style templates, thus combining data selection with presentation.

Graphical icons, including a scrollable box for text, a resizable window for images, and a control box for audio, are assigned to each query and given presentation attributes. The icons are then arranged into a style sheet by either snapping to a grid or by explicitly specifying spatial constraints [8].

The Delaunay^{MM} query language interface supports standard SQL clauses including *select*, *from*, and *where*. It is flexible enough to address queries to distributed relational and object-oriented databases as well as to the Web. In the latter case, an object-oriented model of multimedia documents and elements provides the attributes on which to query. This model extends the HTML 3.2 DTD [22] by incorporating additional metadata attributes, including some from an emerging standard called the STARTS protocol for

*Research supported in part by the National Science Foundation under CAREER Award IRI–9625105.

[1] Delaunay is named after the cubist painter Sonia Delaunay (1885-1979) whose compositions include bright colored geometric figures.

Internet retrieval [15]. Queries to the Web are complicated by its cyclic structure and the fact that the destination for a query is often not known ahead of time (for example, in relational queries the names of the tables storing the sought-after data are supplied by the person forming the query; on the other hand, a query to the Web may or may not include a URL from which to extract the data). Navigational queries, which enable the browsing of document links during query processing, and keyword searches by multiple search engines are therefore supported.

By combining the user-defined style templates with the answers to the queries, a virtual document with pages populated with the retrieved multimedia objects is automatically generated. Each page is associated with one style sheet that determines the layout of the page's elements. Pages are linked together and can be traversed via a *previous/next* mechanism.

The content of each page is based on the answers of the queries associated with it. In some cases, more than one set of page elements (i.e., multimedia objects) may be retrieved in response to a query. The default display specification is to show sets of objects in order of retrieval, with additional sets connected by links that are traversed in a similar manner to page links.

Thumbnail views of each page provide an overview of the entire document, as shown in Figure 1. The style sheet for this document contains a text icon, an image icon, and an audio icon. Queries to populate the icons on each page are first translated into the syntax of the repository to be queried. For example, queries sent to the Web are translated into WebSQL queries [20]. After invoking the queries, further processing of the retrieved objects is performed in order to create the user-specified presentation.

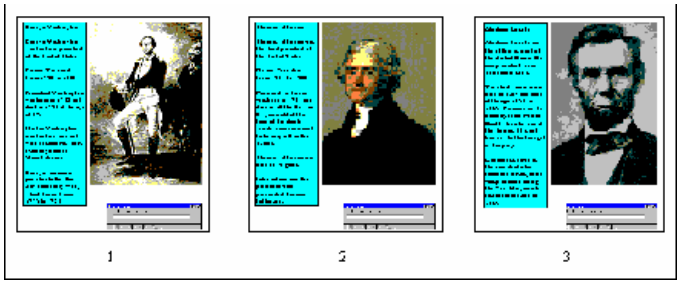

Figure 1: Thumbnail overview of a document.

Within a thumbnail view, pages are arranged in accordance with their position within the generated document, and can be reordered via a drag and drop operation. Selecting a particular thumbnail enlarges that page and makes it the active view.

In addition to the Web, an example of an application domain for which Delaunay^{MM} is currently being implemented is the Perseus Project, a digital library on ancient Greek culture [4]. Knowledge of the data schema, as captured by the data wrapper, provides the attributes on which to query. By providing an integrated query/presentation interface, visitors to the Perseus site [5] will be able to examine the many vases, coins, texts, and other works in ways that are currently not possible. For example, one could display multiple views of one piece of sculpture, compare the same view of many different vases, or arrange a virtual document in which each page represents the artwork of a different artist. In this last case, users could click on one image of a work by a particular artist to view the next work in the set, or could view the works of another artist by clicking on the link to the next page.

Following [9], two types of *save* operations are required to take full advantage of the capabilities inherent in the framework presented here. One saves the actual virtual document for future viewing. The other saves only the query and layout specifications, so that new virtual documents based on previous specifications can be generated, either by editing the specifications or by using more sophisticated mechanisms, such as inheritance and deductive rules (see [6, 7]).

The remainder of this paper is organized as follows. Section 2 describes the Layout and Query Framework, including the Delaunay^{MM} layout and data models. Section 3 contains descriptions of Query Processing and Virtual Document Generation. Our current implementation is described in Section 4, while Section 5 contains a comparison with related work. Our paper concludes with the discussion of future work in Section 6.

2 Layout Specification and Query Formation Framework

Layout specification and query formation are interactive processes by which the user defines (1) the positioning and presentation attributes for the data to be retrieved, and (2) the query criteria specifying what information to retrieve from where. The user can alternate between defining layouts and forming queries, or can complete the layout specification in its entirety before proceeding with the query formation.

2.1 Layout Specification

A user builds a virtual document by creating one or more style sheets. The style sheets define how to present the information to be retrieved from distributed data repositories, including the Web. Having more than one style sheet makes it possible for a single document to incorporate multiple page formats (e.g., title page, chapter pages, and index pages). After entering a label for a style sheet, the user populates the sheet with graphical icons that represent the classes of the multimedia objects, or page elements, to be retrieved. Available icons include a scrollable box for text, a re-sizable window for images, and a control box for audio. Each icon is associated with a particular data repository (e.g., Perseus), multimedia data class (e.g., image), and query (e.g., to retrieve only certain images in that repository).

The simplest way to specify the layout is by snapping to a grid and adjusting the icons to fill the desired space. Each icon will ultimately be replaced by a set of objects that fit the query criteria associated with it. Rather than snapping to a grid, the user can enter numerical values for the dimensional attributes of an icon, such as length and width for a text box, or can place visually specified constraints on the values of those attributes [8]. These constraints are (1) length constraints or (2) overlap constraints (if an object is to be placed on top of another). Length constraints are

linear (unary, binary, or ternary), *maximum* or *minimum* constraints.

Since more than one object within a class may satisfy the query, one can specify how many instances of each class to view at a time by selecting a predefined presentation view. Alternatively, links inherent to a chosen presentation (e.g., stack of cards) can provide the navigational path from one element of the set to the next.

Also assigned to each icon are presentation attributes, such as *font* for text and *color composition* for images, whose values are specified by the user. All instances of the database class that fit the query criteria for an icon are presented throughout the document in accordance with these attributes.

The layout shown in Figure 2 is an example of a style sheet containing an image icon, a scrollable text box, and a non-scrollable text box. Figure 3 shows the query tree associated with that style sheet, which will be described in Section 2.2. In this example, the length and width of the image icon are proportional to those of the largest image that will be contained in that space. These constraints are specified via dialog box options for the length and width attributes. For the non-scrollable text box, the "fill area" attribute has been selected, so that the font and letter size of the text to appear there will be automatically chosen to fill the specified area. A maximum constraint on the height of the page is set to be either (1) the sum of the heights of the image object, the non-scrollable text box, and the space between them, or (2) the height of the scrollable text box, whichever is greater.

In addition to the layout of each page associated with a particular style sheet, the user can organize the overall layout of a virtual document by specifying the relationships between the sets of pages belonging to the different style sheets. Figure 4 shows a layout for a virtual document, or "book" on "Greek Vases" composed with objects resulting from queries to the Perseus database. It depicts a hierarchical organization, with the page associated with the "book cover" style sheet at the top. The level below contains the cover pages for the "chapters" of the book. The pages contained in the chapter on vases found at Harvard University are drawn as children of the Harvard node of the hierarchy. The specification of the layout of virtual documents is achieved using visual rules [11].

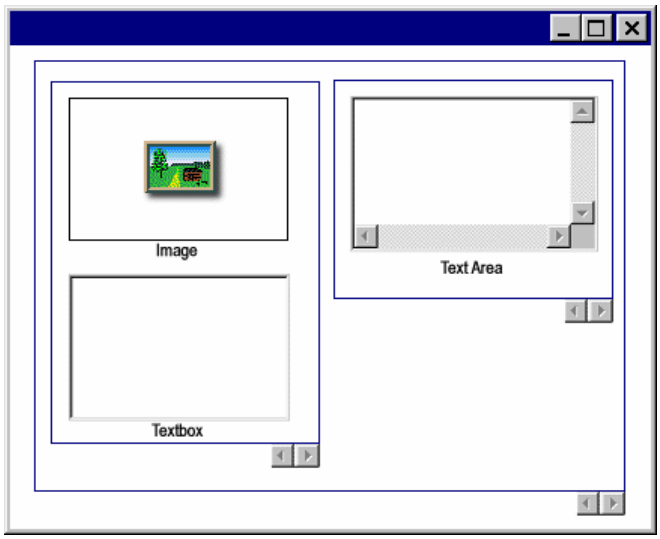

Figure 2: Style sheet.

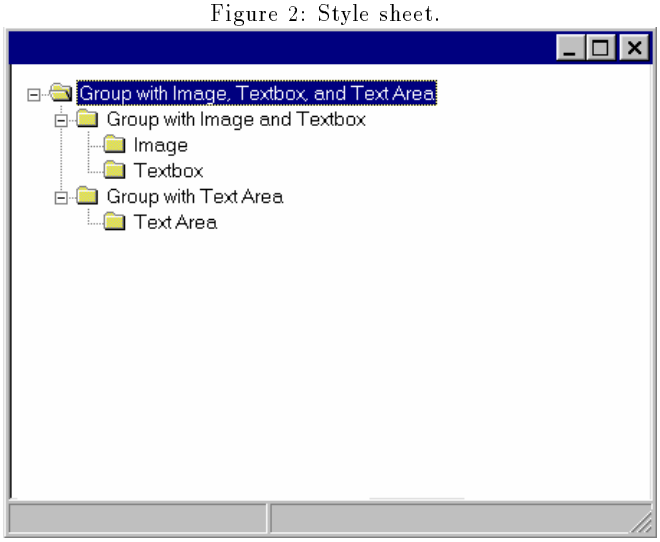

Figure 3: Query tree.

2.1.1 Document Layout Model

An object-oriented model is used to represent the document layout and the data contained within that document. One advantage of using such a model is that each object has a built-in unique identifier, or *oid*. The 17 images associated with Vase 1920.44.54 from Harvard in the Perseus database, for example, are automatically maintained by this model as 17 distinct entities without any programmer input required. In addition, tuples having the same content but representing different real-world entities are distinguishable.

The object-oriented model chosen for representing the document layout and the retrieved data is based on the O_2 [13] and *F-logic* [18] data models. Figure 5 shows the structure of this data model for a virtual document. It has the two primitive type constructors: tuple and set. Syntactically in our representation, tuples are included between square brackets and sets are included within braces.

The *Document* class is defined as a tuple containing *name* and *styles* attributes. The latter is a set valued attribute, since its value is a set of objects of class *Style*. The different objects of class *Style* allow the user to model the different kinds of pages found in virtual documents, as previously described.

As an example, there may be one page with a "Table of Contents" style, and many pages with a "Body" style. Attributes of the *Style* class are (1) *description*, which contains a name (of class *string* given by the user) and (2) *pages*, which contains the set of page objects inheriting the layout defined for a particular style.

The *Page* class has the attribute *elements*, which is set-valued. Each element of the set is a tuple with two at-

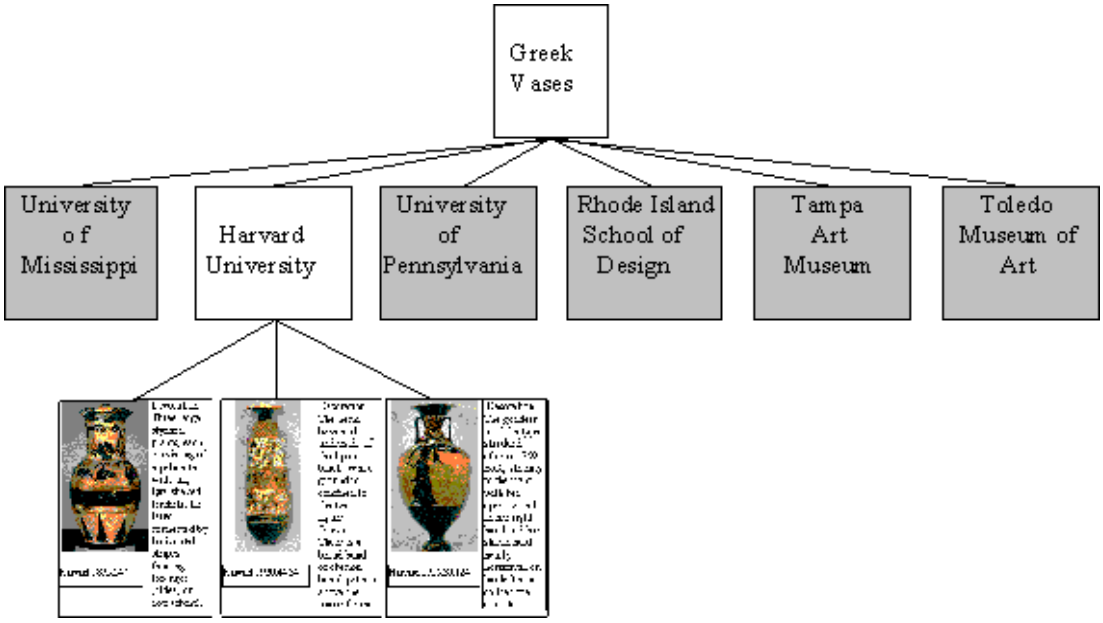

Figure 4: Tree layout of a virtual document.

tributes: *icon_id* and *location*. The value of the latter is a set of coordinates that define the position of the icon within a page. Other attributes of the *Page* class are a reference to the next page, and one to the previous page.

An object of class *Icon* is a tuple made up of a *data* attribute and a *query* attribute. The *data* attribute is associated with a data set (e.g., the set of all images of Greek vases). Each data element in the set has a physical identifier (*pid*) to denote the data repository in which it resides and a value to identify it within that repository. For Web-based data, that value is its URL. A set of data points representing a physical location within an icon is also associated with each data element (e.g., the coordinates of the lower left corner and of the upper right corner of a rectangular region). The *query* attribute stores the query used to populate the icon, as described in the next section.

The multimedia classes of *Text*, *Image*, *Audio*, and *Video* are all subclasses of the *Icon* class. Each inherits the *data* and *query* attributes, and in addition has its own type-specific ones. For example, attributes of class *Text* include *font* and *size*, while attributes of class *Image* include *color content* and *resolution*.

2.2 Query Formation

Query formation is initiated within the layout specification process. Each icon that is added to a style sheet is specified by the user as being part of a query group. The style sheet in Figure 2 shows two "inner" groups, one containing an image icon and a text box icon, and the other containing a text area icon (i.e., a scrollable text box). The "outer" query group contains all three icons. Another closely related structure is represented in Figure 3. A group in this tree representation can be defined by dragging and dropping icons to the tree from the style sheet. Selecting a sub-tree from the query tree highlights the corresponding icons in the style sheet view.

Grouping icons together has both a presentation and a query significance. In terms of presentation, elements of sets associated with one icon are matched with elements of sets associated with the other icons in the group. When the user iterates on a group, the next object in *all* sets within the group is displayed.

Icons within a query group are the values for the *select* portion of the query. Iterating through an inner query group will change only the presentation associated with that particular grouping. Iterating through the outermost query group will change the presentation for the entire page (this is similar to nested loops in a programming language, where the inner loop changes "more quickly" than the outer loop).

An example illustrating the query formation process that uses the Perseus database is the creation of a book of vases from the Harvard Art Museums. The user first creates the style sheet of Figure 2, and places the image and text box (which contains a label associated with that particular view) within one query group, so that the two change together as she iterates through the many different views of each vase. The text area icon, however, is in its own grouping box, because the texts she will be retrieving relate to the vase as a whole. Iterating through these texts should therefore be independent from iterating through the different views of the vase. Finally, all three icons are placed within an outer query group so that she can link from one page to the next, with each page containing information on a different vase within the Harvard collection.

2.2.1 Data Model

After a user has specified a group, the query associated with it can be defined through interaction with the Delaunay^{MM} query interface. In defining the query criteria, the user selects from a list of available attributes and assigns values to them. An object-oriented representation is used in which the

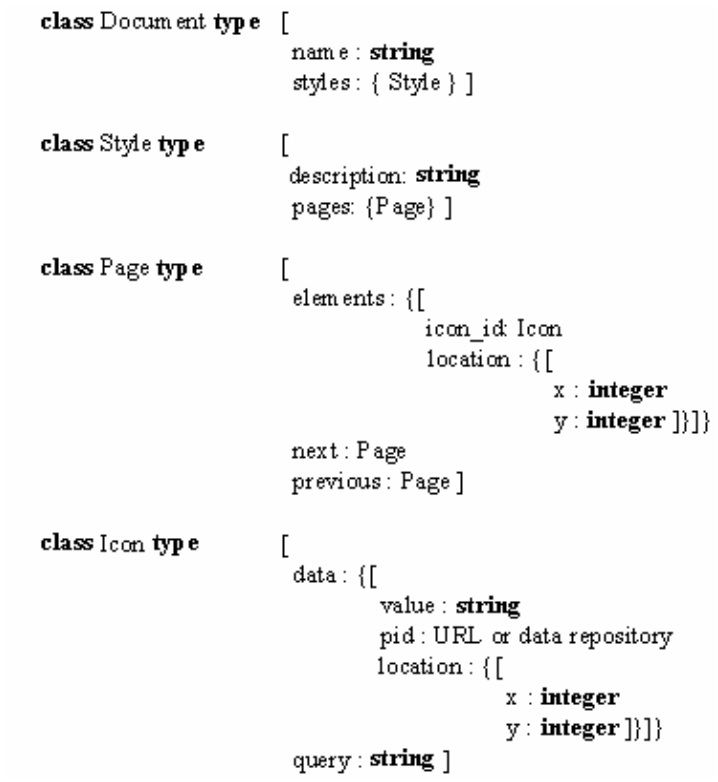

Figure 5: Document layout model.

values of classes and attributes vary depending on the data repositories to be queried. In the case of a database with a known structure, such as that of Perseus, the attributes come directly from the data model and are mapped to an object-oriented representation. For queries to the Web, attributes come from our object-oriented data model of Web documents. Documents are assumed to conform to the syntax of HTML 3.2 [22], which is defined by the combination of the SGML declaration and the document type definition (DTD). The latter declares the element types that are allowed in documents. The names of elements are embedded into HTML documents as tags, which provide directions to browsers on how to display and treat their contents. Attributes appear as additional words within document tags. In the Delaunay^{MM} data model, elements in the DTD have been mapped to classes and element attributes have been mapped to class attributes.

Our data model includes some attributes that are not currently part of the DTD. Most of these have been put into a new *MDATA* class for metadata attributes. Included here are attributes defined by the STARTS protocol for Internet retrieval and search [15]. Namely, the *SRange* attribute relates to the ScoreRange field, and lists the minimum and maximum query scores a document can get within a search engine, while the *AlgID* attribute relates to the RankingAlgorithmID field and identifies the ranking algorithm used for computing scores in that search engine. Once available, both of these attributes could be used for more effective merging of files retrieved by multiple search engines. The *links* attribute, also included in the STARTS protocol, is used for storing all links contained in a file. At the present time, its values are also not provided by search engines.

Other attributes that we added to the *MDATA* class are currently provided by search engines. These include *length*, for the length of the file, and *moddate*, for the last date of file modification. Both of these attributes are also supported by WebSQL [20], the query language into which our Web-destined queries are translated, as explained in Section 2.2.2.

The WebSQL classification of links as *interior* (within the same page), *local* (within the same site), or *global* (outside the current site) has also been added to our model under the *A* class, which is used for describing anchors. The *base* attribute tells the URL of the document containing the link, and the *href* attribute tells the URL of the target of the link.

Figure 6 shows a partial data schema representing the additions we have made to the existing DTD model. Sets of elements, such as the set of URLs represented as {URL}, indicate that zero or more such elements may be present. The symbol "|" is used to represent an OR condition.

2.2.2 Query Language

The Delaunay^{MM} query language interface provides a flexible format in which to enter object-oriented SQL-like queries. These queries are then translated by the Query Processing component (see Section 3) into the syntax accepted by the underlying data repositories. Values for *select*, *from*, and *where* can be chosen from dialog boxes or typed as text. A *mentions* clause for Web queries has been added to support keyword searches by multiple search engines, as provided by [20].

The user first specifies the repositories to be queried, so that the query interface can display the attributes, in scrolling lists, for that repository. The values of the *select* clause are partially specified during the layout specification process: when a text icon is added to a style sheet, Delaunay^{MM} automatically assigns an object identifier (oid) to it, such as "Text1". The user must then select the text attribute to retrieve, such as "title". In the case of an image or an audio recording, the file type to retrieve is specified, such as "gif" for an image, or "wav" for a recording.

To illustrate query formation and the grouping of queries, we will continue with our Perseus example. The relational tables from the Perseus database that are relevant to the queries that follow are shown in Figure 7. In our example, a virtual document of vases from the Harvard collection is being created. A query group contains an image and a text icon. As can be seen in Figure 7, *text* attributes for descriptions of all vases are stored in the Vase table. The Images table contains pointers to images of all objects, including vases, coins, sculptures, etc.

The image of a vase in the query is assigned the oid *Image1* by the system, and the text box is assigned the oid *Text1*. Using scrolling lists and dialog boxes, the user creates the query in Figure 8.

If this were the only query defined for the page, then clicking on the query group's forward and backward links would result in the display of each vase in the Harvard collection along with its name.

By adding a separate query group containing a text box, the user is able to view all the descriptions for each vase.

```
class MDATA type      [
                       src: URL
                       file_type: text/html | text/plain | video/quicktime | ...
                       title: string
                       SRange: score_type
                       AlgID: string
                       length: int
                       moddate: date
                       links: {URL} ]
class A type          [
                       base: url
                       type: interior | local | global
                       href: url
                       label: string ]
```

Figure 6: Search engine classes.

```
TABLE Vase   (Name text                  // key
              Ceramic_Phase text
              Decoration_description text
                  :
              Shape_Description text
              Ware text)

TABLE Coin   (Name text                  // key
              Actual_Weight float4
              Commentary text
                  :
              Reverse_Legend text
              Reverse_Type text)

TABLE Images (Image text                 // key
              Name text                  // foreign key
              Sequence int
                  :
              Caption text
              Location text)
```

Figure 7: Partial schema from Perseus.

```
select   I.Image as Image1, I.Name as Text1
from     I in Images,  V in Vase
where    I.Name contains "Harvard"
         and I.Name=V.Name
```

Figure 8: Image and text group for vases book.

```
select   V.Decoration_Description as Text2
from     V in Vase
where    V.Name contains "Harvard"
```

Figure 9: Text group for vases book.

The query associated with this group is shown in Figure 9.

The last query group contains all of the icons defined for the page, and encompasses the queries shown in the examples of Figures 8 and 9. The user would like each page of the document to contain information on one of the vases in the Harvard collection. The query for the entire page is shown in Figure 10. Each page of the document created from this query refers to a different vase. Within any page, it is possible to iterate through all of the different images of the vase and read the summary information describing it.

In the next example, the user would like to view the two sides (obverse and reverse) of each of the 523 Dewing coin images in the Perseus database. The Images table has a *Sequence* attribute, which is an ordered integer list of the different views stored for each object. Two image icons are added to the style sheet and placed within one query group. The query associated with that group is shown in Figure 11.

```
select  V.Name as Page
from    V in Vase
where   V.Name in
        ((select  I.Image as Image1, I.Name as Text1
          from    I in Images,  V in Vase
          where   I.Name contains "Harvard" and I.Name=V.Name)
        union
         (select  V.Decoration_Description as Text2
          from    V in Vase
          where   V.Name contains "Harvard"))
```

Figure 10: Complete query for vases book.

```
select   O.Image as Image1, T.Image as Image2
from     O in Images, T in Images, C in Coin
where    O.Name = C.Name and O.Sequence = 1 and T.Name = C.Name
         and T.Sequence = 2 and O.Name = T.Name
```

Figure 11: Coin document query.

The result is a document in which each page shows the two sides of every coin with images in the database.

In posing queries to the Web, a particular URL can be specified from which to start the search. Attributes from the Delaunay^{MM} Web file schema appear as selections within scrolling lists. Anchor attributes supporting the interior, local, and global categorizations found in [20] are also available for selection so that the types of links on which to navigate can be specified. Figure 12 shows a query that finds all images of George Washington connected by two or fewer local links to a particular URL, while Figure 13 shows that query as entered into the Delaunay^{MM} query interface to WebSQL.

If the user does not know the starting location for the above query, then a keyword search is needed. All the images connected by two or fewer local links to a Web document containing the keywords "George Washington" are specified. This query is shown in Figure 14. Note that the *where* clause further specifies that the keywords appear in the title (as opposed to, say, anywhere in the document). After the user has fully specified all the queries associated with each style sheet, a "run" option is chosen. The queries are then sent to the Query Processing component. If a query is found to be syntactically incorrect, the user is prompted to edit that query. Additionally, if a "very large number" (as defined by the user) of objects meeting the query criteria are found, particularly in response to Web queries, the user is given the option of reformulating the query to be more specific.

3 Query Processing and Virtual Document Generation Framework

Figure 15 shows the components for Query Processing and for Virtual Document Generation, and how they relate to the other components for Layout Specification and Query Formation, whose functions were described in Section 2.

The Query Processing component is responsible for (1) mapping the schemes of the underlying data repositories to an object-oriented representation for use by the Query Formation component, (2) formatting queries from the Query Formation component into the syntax recognized by their destinations and then executing them, (3) sorting and merging the results of queries, and (4) passing those results on to the Virtual Document Generation component. There, the user-specified layouts are combined with the processed data to form the completed document.

3.1 Query Processing Functions

The Query Processing component is linked via data wrappers to the data repositories. Wrappers export to the Query Processing component: (1) a description of the data types and collections of data, and (2) the types of searches supported by the repositories. In the Query Processing component, the schema description is mapped to an object-oriented representation. Perseus has been implemented using the Postgres database [26], which is relational but supports inheritance of attributes between tables. It is converted to our object-oriented representation using a tool that we have developed [1]. In the case of the Web, attributes from the data model described in Section 2.2.1 are passed to the query interface.

After the queries that define a document have been formed, they are sent to the Query Processing component for translation into a syntax recognized by the query destination. In the case of Perseus, that syntax is SQL. In the case of the Web, queries are translated into WebSQL and then executed by the WebSQL server. The files returned in this latter case go through an additional selection process in which attributes not defined for querying within WebSQL are evaluated. For example, a user might only want a document if a particular phrase appears in one of its headings (as in the query of Figure 14 where the document's title must contain "George Washington"), believing that phrase to be more strongly associated with that docu-

```
select  A.IMG as Image1
from    D in Document such that http://www.loc.gov/washington.html,
        A in Anchor such that linked by D length <= 2
        and A.type = local
where   A.label contains "George Washington" and
        (A.href contains "gif" or A.href contains "jpeg")
```

Figure 12: Web query with a given destination.

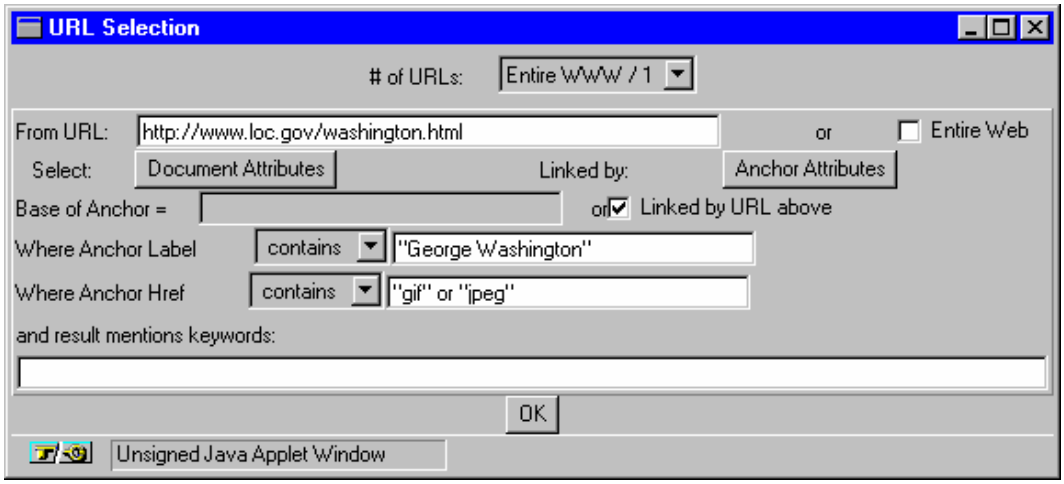

Figure 13: Delaunay^{MM} query interface to WebSQL.

ment than with one in which the phrase only appears in the document's body. This kind of selection is not performed by the WebSQL server. Therefore, we need to parse the HTML documents that have been returned by the WebSQL query and select only those where the phrase appears in the title.

Next, the retrieved data must be merged on the basis of page content as defined by the queries associated with each page. For example, after executing the queries to form a book of vases from the Harvard collection described in Section 2.2.2, the images are matched up with the name of the vase and the text describing them. Figure 16 shows the content, by means of a structured map [12], of the page for the Harvard 1895.247 vase. There are three image-name pairs for this page (the name is required to be shown with each image by the Harvard Museum), and one Decoration_Description.

3.2 Virtual Document Generation

Layout specifications are combined with query results to form the virtual document. The user can browse this document, modify its content and appearance, and save it, as previously described.

3.2.1 Instantiation

The first task performed by the Virtual Document Generation component is the instantiation of each page element with multimedia objects. The retrieved objects are each associated with an icon, a query group, and a page. The graphical constraints that are specified must also be instantiated for each set of objects to which they refer. For example, a length constraint on the height of an image area must be applied to all of the images that will appear in that area.

3.2.2 Evaluation

The Virtual Document Generation component computes the coordinates of the instantiated objects by evaluating the system of instantiated constraints. The constraint resolution strategy consists of building a constraint graph where each vertex represents a constraint variable (i.e., a coordinate), and each edge represents a dependency between two variables established by a constraint. The edge is undirected for linear constraints, and is directed for a maximum or minimum constraint. If the edges of the constraint graph can be oriented so that the resulting digraph has no directed cycles, then we can solve the system of constraints in linear time [10] by evaluating the variables in topological order. Otherwise, we must resort to a general constraint resolution tool (e.g., SkyBlue [24]).

3.2.3 Presentation

The Virtual Document Generation component combines the output from the constraint solving process with the multimedia database objects to form the completed document. Individual page views as well as overall document views are available. A two-way interface extending from the Layout Specification component through the Query Formation component to the Generation component exists so that users can

```
select  A.IMG as Image1
from    D in Document such that D.text mentions "George Washington",
        A in Anchor such that linked by D length <= 2
        and A.type = "local"
where   D.title contains "George Washington" and
        A.label contains "George Washington" and
        (A.href contains "gif" or A.href contains "jpeg")
```

Figure 14: Web query without a given destination.

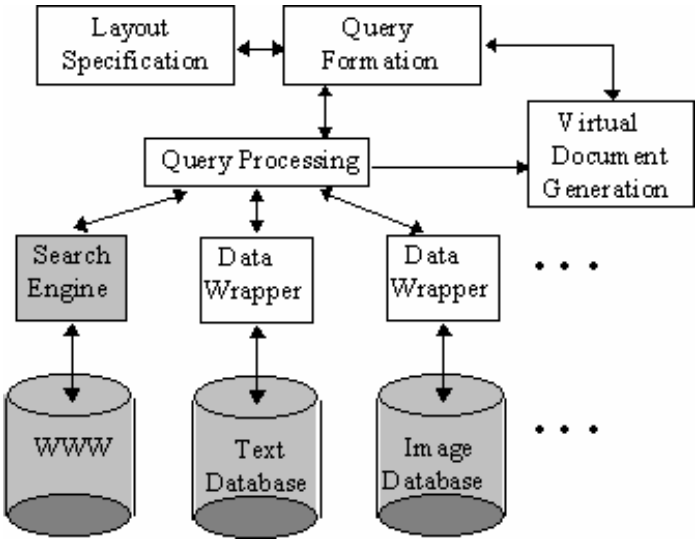

Figure 15: Architecture of Delaunay^{MM}.

make layout-based changes to the virtual document.

4 Implementation

We are currently implementing a system based on the Delaunay^{MM} framework whose architecture is shown in Figure 15. This architecture encompasses the querying and presentation of multimedia data from multiple distributed databases, including the Web. The Presentation Specification and Query interfaces are written in Java so as to be Web-accessible. A client-server architecture, based on standard Java streams and Internet sockets, enables the use of existing constraint resolution tools (e.g., [24]) without re-implementing them in Java.

The Layout Specification component provides the front-end interface through which users define how to present the data to be retrieved. The tool box, as shown in Figure 17, provides icons for adding multimedia element representations to each style sheet. The first five buttons in the top row are used for adding text, images, video, audio, and label elements, in that order, to a style sheet. Once an element has been added, double-clicking on its representation brings up its presentation attributes. The sixth button in that row is used for attaching queries to page elements.

In the second row, the first two buttons are for adding length and overlap constraints. Length constraints can be added between system-defined locations, called "landmarks", on the borders of elements. For example, a distance specification can be set between the center of an image element and the center of a text element by adding a length constraint between those two landmarks. Alternatively, the user can add user-defined location markers called "anchorpoints" to elements by clicking on the third button in this row. This makes it possible to specify length constraints between any two points on two elements, such as the upper left corner of one image and the lower right corner of another. The fourth button is used for viewing and organizing the overall layout of a virtual document, while the fifth button adds a page border that is used for defining page attributes and in setting constraints between the borders of a page and the elements that fall within it. Finally, the sixth button is for a snap-to-grid option. Standard editing functions (e.g., copy, move, delete, and select all) are available from the pull-down menu labeled "Edit".

Figure 18 shows the style sheets window, which contains two templates called "chapters" and "body" that have been created for the virtual document on vases. In the "body" style sheet, the blue line running vertically through the center is a length constraint used for defining the overall size of each page relative to its contents. After a virtual document has been generated, its pages are displayed in a thumbnail view similar in layout to the style sheets window. Clicking on a page makes it the active view.

Two parallel efforts are being pursued at this time in

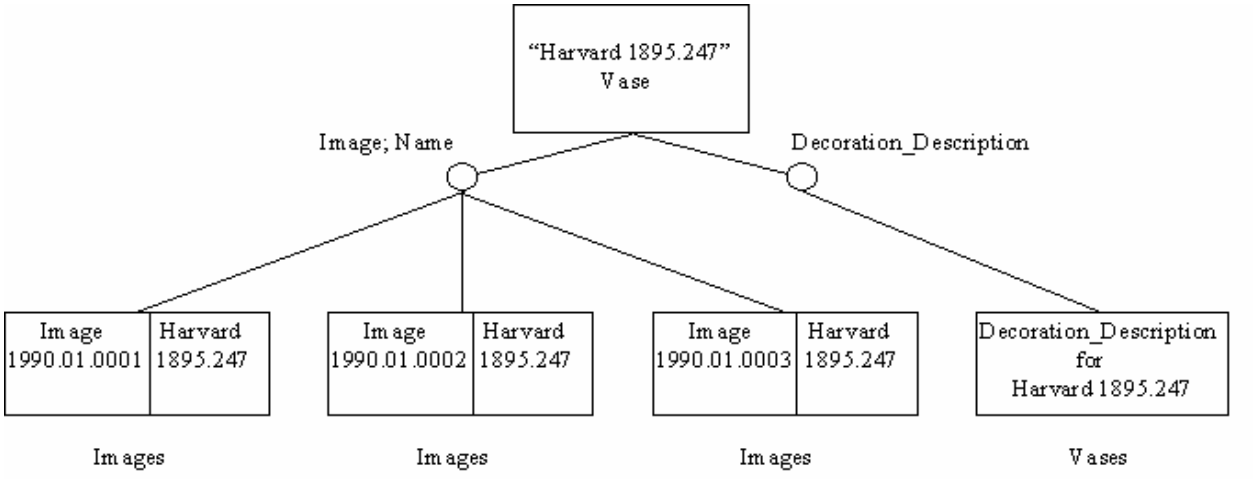

Figure 16: Structured map instance.

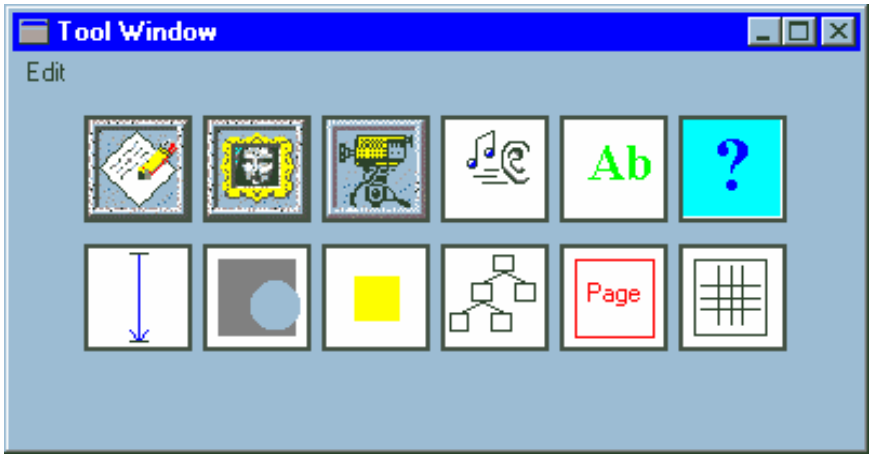

Figure 17: Tool window from Delaunay^{MM}.

terms of interfacing to distributed data repositories. One of these corresponds to queries destined for the Web. Our query interface translates queries into WebSQL, checks for correctness, and sends them to the WebSQL server for processing. The other effort is focused on the Perseus Project. A data wrapper for the Perseus database is currently under development.

5 Related Work

Constraint-based approaches to the automatic generation of multimedia documents include the use of *relational grammars* by Weitzman and Wittenburg [28] and the work on presentation rules by Bertino *et al.* [2].

While in [2, 28] documents are generated from a set of known objects, our approach is designed with external datasets, including the Web, in mind. The work by Weitzman and Wittenburg was an important source of inspiration for the current work. As for the expressiveness of the spatial layout, the work by Bertino *et al.* is quite similar to our former work [6, 7], but differs from it in that it is based on the relational data model. However, they also consider temporal constraints, which we have not yet incorporated into Delaunay^{MM}. In addition, we offer a visual approach that spans from the laying out of the content of individual viewable pages to the modification of features and page orderings found in the completed virtual document.

Other related work includes Garlic [3], DISCO (Distributed Information Search COmponents) [27], and InfoHarness [25] for querying heterogeneous distributed databases. The first of these approaches differs from ours in that they query one database at a time and do not try to integrate data obtained from a variety of sources. While the second approach does incorporate these features, it does not focus on multimedia data and leaves the presentation of retrieved data up to applications programmers. The InfoHarness system uses metadata extraction methods to create information repositories that support run-time access to the original information. While our system incorporates retrieved multimedia objects into user-defined presentations, the information retrieved by InfoHarness is converted to a system-generated combination of HTML forms and hyperlinks, which are then viewed using the Mosaic browser.

The system developed at Xerox PARC [23] uses a variety

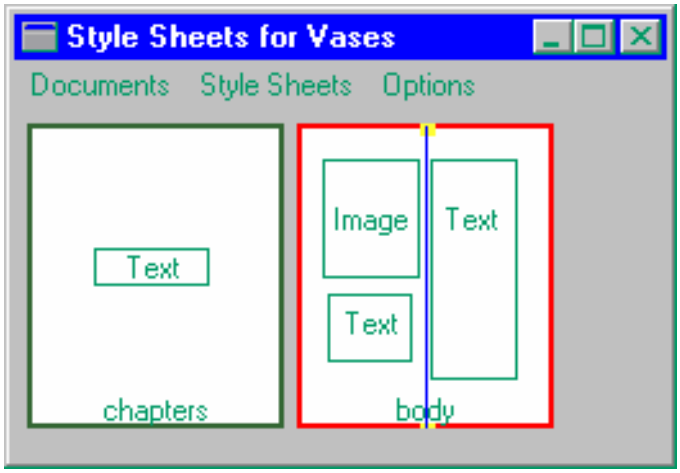

Figure 18: Style sheet window from Delaunay^{MM}.

of 3D displays and integrates an algorithm for the effective browsing of a large collection of documents. Two important differences are our emphasis on user-defined layouts and the availability of our interface over the Web. We have also elected to use 2D displays for faster prototyping and easier access over the Web.

The work by Hüser *et al.* [16] is directed to the generation of documents on the fly. Although this work is intended for the visualization of a single information repository, its presentation objectives are remarkably similar to ours. An interesting difference is that they do not assume pre-defined templates while we have done so, mainly with the objective of simplifying the user's interaction. Using Delaunay^{MM}, the more sophisticated user can, however, achieve similar functionality by using visual rules to shape the layout of the virtual documents [11].

6 Conclusions and Future Work

We have proposed a framework for querying and presenting multimedia information from distributed repositories including the Web. This framework incorporates resource discovery, the retrieval of data as specified by declarative queries, the merging of query results, and the specification of the layout of the retrieved multimedia objects into pages and "books".

In the future, we will expand our access to data repositories other than the Web and Perseus. Examples of other data wrappers and repositories include Garlic [3], QBIC [14], and DISCO [27]. QBIC will allow us to test our ideas on querying images using attributes that are not of type string.

While we have an expressive framework for the specification of spatial layout [10] we have not yet addressed the temporal layout of multimedia components within the virtual documents that are user specified (see for example [19, 29]).

We also plan on conducting usability studies, which are of particular importance to applications intended for a large variety of users. Although the user interface is based on the one in [9], it must support a host of new features related to multimedia data types and distributed data. Our first users are the members of the Perseus Project with whom we have been cooperating. While they fulfill the role of the users who are digital librarians, we would also like to have an experimental site available to the casual users of the Perseus site [5]. Given the popularity of this site, we believe that it would be an ideal testbed for our ideas.

Acknowledgments

We would like to thank Greg Crane, Maria Daniels, and David Smith from the Perseus Project for useful discussions. The vase photos by Maria Daniels are courtesy of the Harvard University Art Museums.

References

[1] M. Averbuch and I. F. Cruz. From Relational to Object-Oriented Databases: Migration Algorithm and Software, April 1996. Manuscript available at http//www.cs.tufts.edu/~averbukh/proj2.html.

[2] E. Bertino, B. Catania, E. Ferrari, and A. Trombetta. Presentation Constraints for Multimedia Data. In *Intl. Workshop on Multimedia Information Systems*, pages 26–28, 1996.

[3] W. F. Cody *et al.*. Querying Multimedia Data from Multiple Repositories by Content: the Garlic Project. In *3rd IFIP Working Conference on Visual Database Systems*, 1995.

[4] G. R. Crane, ed. Building a Digital Library: the Perseus Project as a case Study in the Humanities. In *First ACM International Conference on Digital Libraries*, pages 3–10, 1996.

[5] G. R. Crane, ed. The Perseus Project, May 1997. http://www.perseus.tufts.edu.

[6] I. F. Cruz. DOODLE: A Visual Language for Object-Oriented Databases. In *ACM-SIGMOD Intl. Conf. on Management of Data*, pages 71–80, 1992.

[7] I. F. Cruz. User-defined Visual Query Languages. In *IEEE Symposium on Visual Languages (VL '94)*, pages 224–231, 1994.

[8] I. F. Cruz. Expressing Constraints for Data Display Specification: A Visual Approach. In V. Saraswat and P. V. Hentenryck, editors, *Principles and Practice of Constraint Programming*, pages 443–468. The MIT Press, 1995.

[9] I. F. Cruz, M. Averbuch, W. T. Lucas, M. Radzyminski, and K. Zhang. Delaunay: a Database Visualization System. In *ACM-SIGMOD Intl. Conf. on Management of Data*, pages 510–513, 1997.

[10] I. F. Cruz and A. Garg. Drawing Graphs by Example Efficiently: Trees and Planar Acyclic Digraphs. In *Graph Drawing '94*, number 894 in Lecture Notes in Computer Science, pages 404–415. Springer Verlag, 1995.

[11] I. F. Cruz and W. T. Lucas. Delaunay^{MM}: a Visual Framework for Multimedia Presentation. In *IEEE Symposium on Visual Languages (VL '97)*, 1997.

[12] L. M. L. Delcambre, D. Maier, R. Reddy, and A. L. Structured Maps: Modeling Explicit Semantics over a Universe of Information. *International Journal of Digital Libraries*, 1(1):20–35, 1997.

[13] Deux *et al.*. The Story of O_2. In F. Bancilhon, C. Delobel, and P. Kanellakis, editors, *Building an Object-Oriented Database System (The story of O_2)*. Morgan Kaufmann Publishers, San Mateo, California, 1992.

[14] M. Flickner et al. Query by Image and Video Content: the QBIC System. *IEEE Computer*, 28(9):23–32, 1995.

[15] L. Gravano, C.-C. Chang, H. Garcia-Molina, and A. Paepcke. STARTS: Stanford Proposal for Internet Meta-Searching. In *ACM-SIGMOD Intl. Conf. on Management of Data*, pages 207–218, 1997.

[16] C. Hueser, K. Reichenberger, L. Rostek, and N. Streitz. Knowledge-based Editing and Visualization for Hypermedia Encyclopedias. *Communications of the ACM*, 38(4):49–51, April 1995.

[17] J. Foley and J. Pitkow, eds. Research Priorities for the World Wide Web, October 1994. Authors: R. C. Berwick, J. M. Carroll, C. Connolly, J. Foley, E. A. Fox, T. Imielinski, and V. S. Subrahmanian; manuscript available at http://www.cc.gatech.edu/gvu/nsf-ws/report/Report.html.

[18] M. Kifer, G. Lausen, and J. Wu. Logic Foundations of Object-Oriented and Frame-Based Languages. *J. ACM*, 42(4):741–843, July 1995.

[19] M. Y. Kim and J. Song. Multimedia Documents with Elastic Time. In *The Third ACM International Multimedia Conference*, pages 143–154, 1995.

[20] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *International Journal of Digital Libraries*, 1(1):54–67, 1997.

[21] B. A. Myers, J. D. Hollan, and Isabel F. Cruz, *eds.* Strategic Directions in Human Computer Interaction. *ACM Computing Surveys*, 28(4), 1996.

[22] D. Raggett. HTML 3.2 Reference Specification, W3C Recommendation 14-Jan-1997. Available from http://www.w3.org/pub/WWW/TR/REC-html32.html.

[23] R. Rao *et al.* Rich Interactions in the Digital Library. *Communications of the ACM*, 38(4):29–39, April 1995.

[24] M. Sannella. The SkyBlue Constraint Solver. Technical Report 92-07-02, Computer Science Department, University of Washington, February 1992.

[25] L. Shklar, A. Sheth, V. Kashyap, and S. Thatte. InfoHarness: The System for Search and Retrieval of Heterogeneous Information. In *Proceedings of ACM SIGMOD*, 1995.

[26] M. Stonebraker, L. Rowe, and M. Hirohama. The Implementation of POSTGRES. *Journal of Data and Knowledge Engineering*, 2(1):125–142, March 1990.

[27] A. Tomasic, R. Amoroux, P. Bonnet, O. Kapistkaia, H. Naacke, and L. Raschid. The Distributed Information Search Component (DISCO) and the World Wide Web. In *ACM-SIGMOD Intl. Conf. on Management of Data*, pages 546–548, 1997.

[28] L. Weitzman and K. Wittenburg. Automatic Presentation of Multimedia Documents using Relational Grammars. In *ACM Multimedia Conference*, 1994.

[29] P. Zellweger. Temporal layout. In I. F. Cruz, J. Marks, and K. Wittenburg, editors, *Effective Abstractions in Multimedia: Layout, Presentation, and Interaction (Proc. of the ACM Workshop, in association with the ACM Multimedia Conference '95)*. San Francisco, Ca., November 1995. Available at http://www.cs.tufts.edu/~isabel/mmwsproc.html.