Delaunay^{MM}: a Visual Framework for Multimedia Presentation*

Isabel F. Cruz
Department of Computer Science
Worcester Polytechnic Institute
ifc@cs.wpi.edu

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

Abstract

We introduce a visualization framework called Delaunay^{MM} for querying and presenting multimedia information in distributed databases. Users can tailor the layout of virtual documents containing multimedia information at the same time as they specify their contents in an intuitive yet formal way, using a Web-available interface. The data model that we propose is object-oriented, and the query and presentation languages are declarative and incorporate layout constraints. Our framework encompasses the retrieval of heterogeneous distributed information whose data model does not need to conform to the one used by Delaunay^{MM}.

1 Introduction

A database management system (DBMS) enables the storage of large amounts of data that are presented back to the user in the easily understood formats of tables or spreadsheets.

Two recent technological trends have had a major impact on the way data is managed and viewed:

- The advent of multimedia data types makes traditional display and reporting formats no longer applicable. Because of the richer types, querying also goes beyond the capabilities of the traditional relational and object-oriented database languages.
- The increased accessibility to distributed information repositories by virtually any user with a modem-equipped computer means that the information may be located anywhere on the Web and that answers to queries may have to be extracted from more than one information repository.

Therefore, there is the need to define a new paradigm that allows users to view and create presentations from distributed sources in much the same way that a traditional DBMS lets users view and create tables and reports. Likewise, visual query languages that are well-developed for traditional DBMSs [4] need to be adapted to the new kinds of interaction provided by distributed multimedia data. The challenge for visualization researchers is to provide "effective" tools that enable a wide variety of users to search, browse, select, and customize information from these sources [10, 14, 17].

Specifically, it should be possible to:

1. Query data from multiple repositories for interactive viewing.
2. Specify the format and presentation of composite database views.
3. Combine query results from (1) with view specifications from (2) to automatically generate virtual documents.
4. Customize presentation attributes of individual media elements (i.e., text, image, video, and audio objects).

The framework we propose is based on a visual query and presentation language for specifying multimedia database views and automatically generating user-defined virtual documents. Delaunay^{MM} is a multimedia extension to the *Delaunay Database Visualization System* [11], an interactive, constraint-based system for visualizing object-oriented databases. Users create visual programs, by arranging geometric graphical objects and graphical constraints to form a "picture" that specifies how to visualize data objects belonging to a database class.[1]

The visual programs pictorially specify in an intuitive yet formal way the visualization of database objects. By mapping objects in the database to graphical objects, users can specify both the selection criteria and the presentation of the queried data. Delaunay supports graphical transformations of data to user-defined representations, including standard representations such as pie and bar charts [7, 8]. These

*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.

layouts enhance the user's ability to interpret trends and facts that are not readily perceivable in tabular form.

However, while Delaunay allows for the querying and visualization of a single traditional object-oriented database, Delaunay^{MM} is geared to distributed multimedia databases. Delaunay^{MM} is built upon a database schema that defines a virtual document as being a sequence of viewable "pages", each of which contains a composition of multimedia objects retrieved from distributed sources with a variety of data formats. The user forms a query for specifying the data that populates the document and creates a style template showing how to present the retrieved data.

The remainder of this paper is organized as follows. We give an overview of Delaunay^{MM} in Section 2. The document layout data model is presented in Section 3. In Section 4, the query formation, layout specification, query processing, and virtual document generation aspects of the Delaunay^{MM} framework are described. Our implementation efforts are outlined in Section 5. A comparison with related work is contained in Section 6, and Section 7 concludes with a discussion of directions for future work.

2 Overview of Delaunay^{MM}

In a *virtual document*, the class of each multimedia object, or *page element*, to be retrieved is visually designated through the use of *graphical icons*, which include a scrollable box for text, a resizable window for images, a control box for audio, etc. Each icon is therefore associated with a particular multimedia data class (e.g., image), data repository (e.g., URL) and query (e.g., to retrieve only certain images).

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

Icons are arranged into a page layout by either snapping to a grid or by explicitly specifying spatial *constraints* [9]. Figure 1 shows an example of a user-defined page layout.

After configuring a page view and initiating the query process, a virtual document composed of multiple pages is subsequently created by populating the document layout specification with the retrieved multimedia objects. Each page is associated with one style sheet that determines the layout of the page's elements. A generated page is therefore an instance of class *Page*, which is a subclass of class *Style*. The attribute values of page elements can be modified for the entire document as well as on a page-by-page basis. Pages are linked together and can be traversed via a *previous/next* mechanism.

Thumbnail views of each page provide an overview of the entire document, as shown in Figure 2. Pages

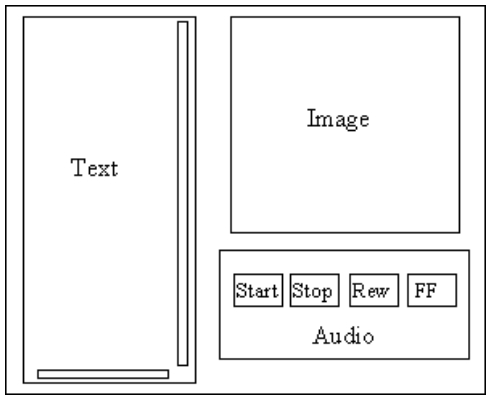

Figure 1: Layout specification.

are arranged here 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.

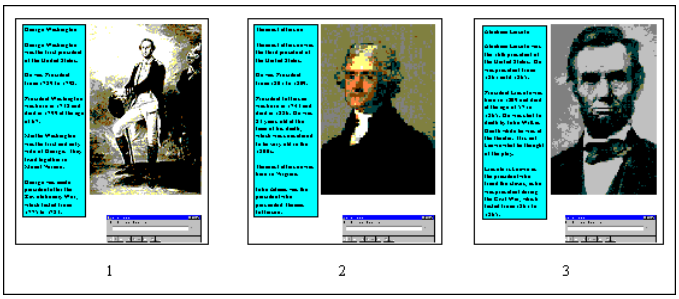

Figure 2: Virtual document layout.

The content of each page is based on the results of the user-specified queries associated with groups of one or more icons. In some cases, a query may result in more than one page element being retrieved for any given page. Each icon therefore represents a set of retrieved objects, or class instances. The default display specification is to show the first object retrieved, with additional objects viewable through user selections. Alternatively, the selection of a *presentation format* permits the user to specify how many instances of a class to display at a time.

As an example of a virtual document, a user might pose a query to select, for each past and current president of the United States, an image from an Image Database and a biography from a Text Database and, on each page of the document, show one image and one biography per president. The user associates (1) the image icon with the Image Database and with the image portion of the query, and (2) the text icon with the Text Database and with the text portion of the query. The first page of the generated document shows a picture of George Washington along with his biography, but there are actually three retrieved images and two

biographies. To view the second image retrieved, the user selects an option to view the next page element associated with the image icon, and so on.

As in [11], to take full advantage of the capabilities inherent in the framework presented here, two types of *save* operations are required. 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 visual rules (see Section 4.2 and [8]).

3 The Document Layout Data 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*. Thus the three pictures of George Washington in the prior example are automatically maintained by this model as three distinct entities without any programmer input required. Also, two tuples with the same content but different real-world entities are distinguishable.

Another advantage of this model is that classifying data together according to shared characteristics is a natural and efficient method for dealing with large volumes of complex data.

The object-oriented model chosen for representing the document layout and the retrieved data is based on the O_2 [2] and *F-logic* [16] data models. It has 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 with a *name* attribute and a *styles* attribute that contains a set of style objects. Objects of class *Style* allow the user to model the different kinds of pages found in a virtual document, as previously described. For 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 a user-specified *description* and *pages*, which are the set of page objects inheriting the layout defined for a particular style. While graphical icons are used to represent the layout of a style object, instances of those icons make up the layout of each page and are therefore associated with each object of class *Page*. Other attributes of the *Page* class are a reference to the next page, and one to the previous page.

An icon is made up of a *data* set (e.g., the set of all the pictures of George Washington) and the *query* used to populate it. There are also attributes specific to the icon data type (see Figure 4). Each data element in the data set has a physical identifier (*pid*) to denote the data repository in which it resides and a value (e.g., its identifier within the data repository). For Web-based data, the pid 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). Figure 3 shows the structure of the data model for a virtual document.

```
class Document type [
                    name: string
                    styles: {Style} ]


class Style type [
                 description: string
                 pages: {Page} ]

class Page type [                                          10
                 elements:
                   {[
                      icon_id: Icon
                      location : {[
                                    x: integer
                                    y: integer ]} ]}
                 next : Page
                 previous : Page ]

class Icon type [                                          20
                 data:
                  {[
                     value: string
                     pid: URL
                     location: {[
                                  x: integer
                                  y: integer ]} ]}
                 query: string ]
```

Figure 3: Description of the document layout model.

The data schema showing representations for the *Text, Image, Audio*, and *Video* classes found in the Delaunay^{MM} data model is shown in Figure 4. These classes are all subclasses of the *Icon* class found in Figure 3. For clarification purposes, options for attribute values may appear in parentheses. The attributes included here are not meant to be an exhaustive list, but rather to provide an overview of the model. Because it is object-oriented, the data model can easily be extended.

4 The Delaunay^{MM} Framework

The components that serve as the basis for our framework are shown in Figure 5. Delaunay^{MM} interfaces to both local and Web databases via data wrappers [5].

4.1 Query Formation

A virtual document is generated from data retrieved in response to user-defined queries. Each query is associated with a group of one or more icons; all queries taken together define the search criteria for the entire virtual document.

```
class Text type [
        format: string      (txt, html, doc, tex, ...)
        title: string
        content: string
        keywords:  {[
                word: string ]}
        font: string
        size: integer
        color: string
        language: string ]

class Image type [
        format: string     (gif, jpeg, ...)
        title: string
        color_content: string  (B&W or color)
        size:  [
                units: string  (pixels or inches)
                length: integer
                width: integer]
        resolution: float ]

class Audio type [
        format: string     (wav, au, midi, ...)
        title: string
        length_of_play: float]

class Video type [
        format: string     (AVI, Quicktime, ...)
        title: string
        color_content: string (B&W or color)
        frames: integer
        size:  [
                units: string  (pixels or inches)
                length: integer
                width: integer]
        resolution: float
        transfer_rate: integer (frames per second)
        length_of_play: float ]
```

Figure 4: Object-oriented schema for the retrieved data.

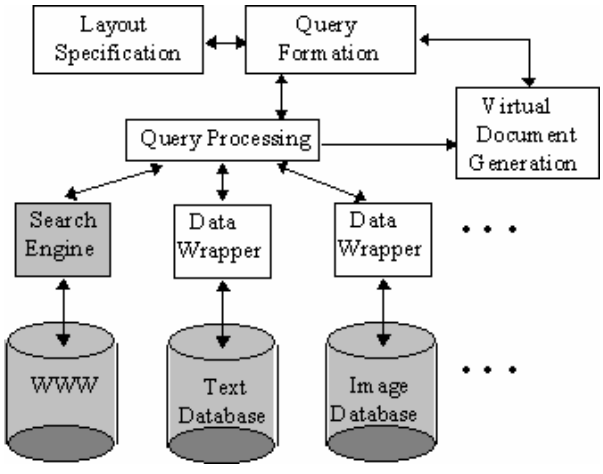

Figure 5: Architecture of Delaunay^{MM}

The query language associated with this structure is an object-oriented extension of the SQL language inspired by the languages proposed for the O_2 data model, namely O_2SQL [2], and for the F-logic data model, namely XSQL [15].

In posing a query, the user provides two components: (1) the search criteria defining the attributes of the multimedia objects to retrieve, and (2) a query destination. The latter may be selected from local databases or may be one or more addresses on the Web, as further explained in Section 4.3.

As a simple example, suppose one wished to find the titles of all documents written in English about George Washington. An appropriate destination would be the home page on the Web for the Historical Collections of the National Digital Library provided by the Library of Congress. Such a query is represented in Figure 6. We use a standard notation where, for example, `X.title` denotes the value of the attribute `title` for the object variable `X` that, in this example, ranges over all the objects of class `Text`.

```
select      X.title
from        Text X
where       X.title contains "George Washington"
            and X.language = "English"
destination http://rs6.loc.gov/ammem/ammemhome.html
```

Figure 6: SQL-like query.

Thus, Query Formation refers to the process of specifying the selection criteria and search destination that will ultimately determine the content of the virtual document to be generated. How the retrieved data is to be presented back to the user is the domain of the Layout Specification component, as described next.

4.2 Layout Specification

Each user-defined query is assigned to a group of one or more graphical icon that reflects the type of data to be retrieved. In the prior example, the query for images of presidents and for audio recordings related to those presidents has an Image icon and an Audio icon associated with it. These icons are arranged into a page layout by the user.

The simplest way to specify the layout is by snapping to a grid and adjusting the icons to fill the desired space. Implicitly, this defines constant length constraints between the icons and the page borders. *Length constraints* denote the distance between two points (horizontal, vertical, or Euclidean).

Presentation formats can also be used to specify how many instances of each class to view at a time; links inherent to the chosen presentation (e.g., stack of cards) provide the navigational path from one grouping to the next.

Often, however, the user cannot know exactly how much space to assign to each element on a page because it will depend upon the characteristics of the retrieved elements. In these cases, the layout is speci-

fied using variable length constraints. We use a class of basic constraints over a set of variables, where linear arithmetic expressions are combined with *min/max* operators (e.g., to set the height of a page to the maximum height of the objects it contains). The variables may refer to other constraints in the same page layout, or to the value of a numerical database attribute (so that we can set a length to be proportional to that attribute value).

The layout shown in Figure 7 is an example of some of the ways in which constraints can be used to specify page layout. The description of the constraints that follows is not meant to include all available options but rather to serve as a representative sample.

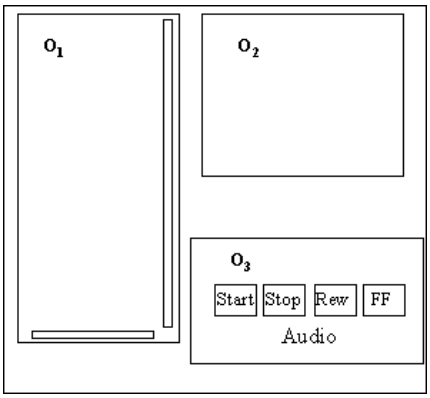

Figure 7: Layout specification.

Three types of objects, each bearing a label for identification purposes only, are specified in Figure 7. o_1 is a text object. The user has a particular size in mind for o_1, and therefore sets its height and width by snapping to a grid (resulting in length constraints that are invisible to the user). She also selects the "Fill Area" mode, which will calculate the font and letter size to use so that the text will fill the specified area. For object o_2, however, which is an image object, she wants the length and width to be equal to those of the largest image retrieved. She therefore sets those constraints to be proportional to the maximum value of the length and width attributes of the retrieved images. Length constraints are also used to specify the minimum allowable width of the space between o_1 and o_2, and the minimum height of the space between o_2 and o_3. Finally, the height of the page is set using a *max* constraint to be either the height of o_1, or to be the height of o_2 plus o_3 plus the space between them, whichever is greater.

The *Style* class is the building block for our layout model. However, the overall layout of the set of pages related to a particular style (e.g., that compose a virtual document) should also be customizable by the user. Figures 8 and 9 show two layouts for a virtual document on "World Presidents". Figure 8 depicts a hierarchical organization, with the "book cover" at the top. The level below contains the cover pages for the "chapters" of the book. The pages contained in the USA chapter are drawn as children of the USA node of the hierarchy.

Another representation for this virtual document is shown in Figure 9. In this case, a *containment layout* is shown, where all the chapters are drawn inside a larger labeled rectangle representing the book. Likewise, the pages that make up the USA chapter are shown inside a labeled rectangle depicting this chapter.

We consider the scenario where the hierarchy view has been defined, and the user would like to map this representation to a containment view. In this case, the pages have to be rearranged. The user composes a visual rule as shown in Figure 10. This rule is to be read right to left (like in Prolog), as a logical implication. The meaning of this rule can be described as follows:

1. If there is an edge from a page at a higher level of the hierarchy to another page, then the contents of the child pages are to be drawn inside of a rectangle labeled by the name of the parent page. The grouping of the child pages is represented by the dotted rectangle. This rule applies both to the book and chapters within the book and to each chapter and pages within the chapter.

2. The layout of the pages inside of the container page also has to be defined. In this case we want to specify that they should be drawn as a horizontal list. The page that goes to the right of the first page is the one referred to by *"next"*.

If the format of the pages containing information about the presidents is not to be changed, then the user does not need to compose another rule. By default, these pages will remain unchanged since the previous rule does not apply to their contents. If, on the other hand, the style of these pages needs to be changed, a rule can be written to reorganize the information in each of these pages as in Figure 11.

The relations between pages in a virtual document that are inherent to the Delaunay^{MM} framework, the constraint-based relations between page elements, and the mapping rules can be formally modeled using F-logic [16] by extending [7].

4.3 Query Processing

The Query Processing component is responsible for:

1. Obtaining information about the underlying data repositories from the data wrappers to which it interfaces.

2. Providing the user with that information via the Query Formation component.

3. Sending each user-defined query to the appropriate data wrapper, where it is translated into

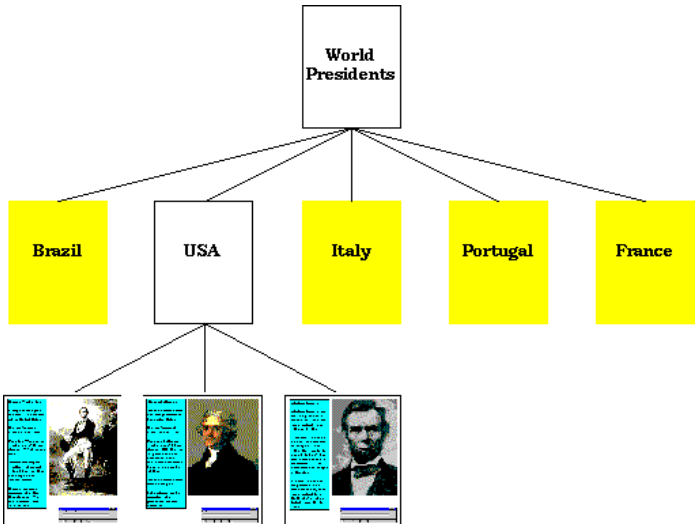

Figure 8: Tree layout of a virtual document.

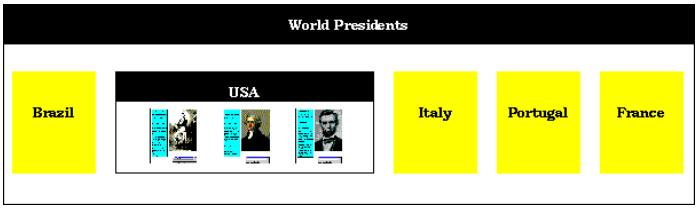

Figure 9: Containment layout of a virtual document.

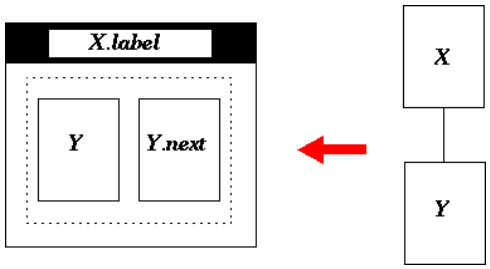

Figure 10: Visual rule for the mapping from the tree to the containment layout.

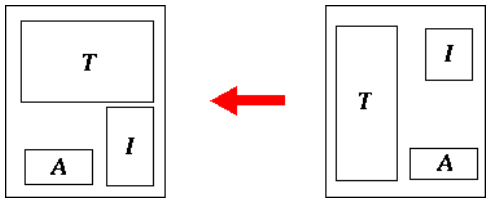

Figure 11: Visual rule for changing the layout of each page.

the syntax recognized by the data repository for which it is intended.

4. Collecting and merging the results of a user's queries.

5. Passing those results on to the Virtual Document Generation component.

As an example, in forming a query to an O_2 object-oriented database, the user is provided with a list of objects along with their associated attributes. Selection criteria are attached by the user to one or more of those attributes. The resulting query is translated from the Delaunay^{MM} object-oriented syntax into O_2SQL [2] by one of the wrappers shown in Figure 5 and is then processed by the O_2 repository. Query results are merged by the Query Processing Component with any other results obtained from queries defined for the same style sheet and are sent on to the Virtual Document Generation component for final processing.

The protocol for queries to the Web differs slightly because each query is sent to multiple search engines, necessitating the removal of duplicated information. In addition, because search engines only support keyword searches, a "second pass" is required to fully process all the query criteria specified by the user. Finally, the desired multimedia component, such as images or text, must be extracted from the retrieved files.

4.4 Virtual Document Generation

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

There are three functions performed by the Virtual Document Generation module:

1. It instantiates each page with the multimedia objects returned from query processing.

2. It computes the coordinates of the instantiated objects by evaluating the system of instantiated constraints.

3. It presents the virtual document to the user for further modification/viewing/storage.

4.4.1 Instantiation

In performing the instantiation, the returned multimedia objects are examined in two passes. In the first pass, we identify the retrieved database objects, and associate each with an icon and a page. We also instantiate the graphical icons and variables for the coordinates of the points used in defining constraints. In the second pass, we instantiate the constraints over the previously instantiated variables.

4.4.2 Evaluation

The Delaunay^{MM} framework supports the following types of length constraints. Here the w_i's and z_i's

denote variables, and the remaining symbols denote constants.

- $w_1 = b$; (unary constraint)
- $a_1w_1 + a_2w_2 + b = 0$; (binary constraint)
- $a_1w_1 + a_2w_2 + a_3w_3 + b = 0$; (ternary constraint)
- $z_1 = \min(z_1, a_1w_1+a_2w_2+a_3w_3+b, c_1w_1+c_2w_2+c_3w_3 + d)$; (min constraint)
- $z_1 = \max(z_1, a_1w_1+a_2w_2+a_3w_3+b, c_1w_1+c_2w_2+c_3w_3 + d)$; (max constraint)

The constraint resolution strategy is as follows. We build 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 a unary, binary or ternary constraint, and is directed (from w_i to z_1) for a max or min constraint. If the edges of the constraint graph can be oriented such that the resulting digraph has no directed cycles, then we can solve the system of constraints in linear time [12] by evaluating the variables in topological order. Otherwise, we should resort to a general constraint resolution tool (e.g., SkyBlue [19]).

4.4.3 Presentation

In this final step, the Virtual Document Generator combines the output from the constraint solving process with the multimedia database objects to form the completed document. The individual page view as well as the overall document view are available to the user.

A two-way interface between the Layout Specification and the Virtual Document Generation components exists so that users can make layout-based changes to the virtual document.

5 System Implementation

In this section we briefly list some of the main characteristics of of the Delaunay^{MM} system, which we are currently implementing. Similar techniques are already implemented in [11].

User Interface The interface gives an overview of each virtual document and attempts to minimize the amount of information on the screen. The user interface has access to local databases to prompt the user with their names, class names, and attribute names as appropriate, and provides access to other saved documents and specifications through a simple pull-down menu. The drawing pad implements standard editing functions (e.g., cut, paste, move, group). The interface is implemented in Java, and is therefore multiplatform. Furthermore, it is available on the Web, and hence can be accessed by any Java-enabled browser. The output pad that displays the resulting virtual document (see, for example, Figures 2, 8 and 9) allows for zooming and for the display of the attributes associated with each displayed object (e.g., its URL).

System Modularity The Delaunay^{MM} system was designed for extensibility: we defined completely the languages between every two communicating software modules, so as to allow for future replacement or addition of other modules. A client-server architecture, based on standard Java streams and Internet sockets, enables the use of existing constraint resolution tools (e.g., SkyBlue [19]) without reimplementing them in Java.

Expressiveness and Efficiency The expressiveness of our presentation framework stems from the capability to express recursion (which is useful, e.g., to display recursive data types), from the mapping rules, and from the variety of constraints that can be expressed [7, 9]. When solving constraints, one of the main concerns is to provide algorithms for solving them. It is usually the case, however, that the more general the constraints, the more inefficient the constraint solver will be. Therefore, the challenge is to provide constraints that can express a wide variety of displays and can also be solved efficiently. In our system, the constraint systems associated with a significant class of visualizations can be solved in linear time [12].

6 Related Work

Constraint-based approaches to the automatic generation of multimedia documents include the use of *relational grammars* by Weitzman and Wittenburg [20] (which was an important source of inspiration for the current work) and the work on presentation rules by Bertino *et al.* for relational databases [3]. While in [3, 20] documents are generated from a set of known objects, our approach is designed with external datasets, including the Web, in mind. Among the other features that distinguish our framework from [3, 20] is the specification of constraints that are proportional to database attributes and the WYSIWYG approach based on a grid layout, which facilitates the specification of many spatial constraints. Also, we support 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. However, we do not consider temporal constraints, which are proposed in [3].

Other related systems include [5] and the work at Xerox PARC [18]. The former approach emphasizes database querying but does not focus on the interface or presentation aspects. On the other hand, the system in [18] uses a variety 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 2D displays for faster prototyping and easier access over the Web.

The work by Hüser *et al.* [13] 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 the visual rules to shape the layout of the virtual documents as partially illustrated in Section 4.2 (see also [7, 8]).

7 Future Work

We are concentrating on the following aspects:

Database Integration In the current implementation, only databases stored on the server side can be visualized. In the future, we will support access to databases stored on the client side and to databases available on the Web. Furthermore, we are developing an interactive tool to migrate relational databases into object-oriented databases [1], thus providing the functionality of a data wrapper for accessing relational databases.

User Interfaces and Usability In addition to the style-based interface described above, we are implementing graphical user interfaces using Java to assist users in resource discovery on the Web, and for database integration, visualization of the Web, and visual languages for interfacing Web search engines (e.g., AltaVista). We plan to conduct usability studies, which are important for applications intended for a large variety of users. Such studies may help in determining other kinds of queries that we have not yet considered as well as in defining new presentation possibilities.

Applications We are looking at concrete applications, such as the Perseus Project, a digital library on ancient Greek culture [6] with a known data structure. By providing an integrated query/presentation interface, visitors to the Perseus site 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.

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] F. Bancilhon, S. Cluet, and C. Delobel. A Query Language for 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.

[3] 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.

[4] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual Query systems for Databases: A Survey. *Journal of Visual Languages and Computing*, March 1997.

[5] 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.

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

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

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

[9] 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.

[10] I. F. Cruz. Tailorable Information Visualization. *ACM Computing Surveys*, 28A(4), 1996.

[11] 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*, 1997.

[12] 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.

[13] 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.

[14] 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.

[15] M. Kifer, W. Kim, and Y. Sagiv. Querying Object-Oriented Databases. In *ACM-SIGMOD Intl. Conf. on Management of Data*, pages 393–402, 1992.

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

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

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

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

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