VQBD: Visualizing, Querying, and Browsing Semistructured Data[*]

Sudarshan S. Chawathe Thomas Baby Jihwang Yeo

Computer Science Department, University of Maryland, College Park, Maryland 20742

{chaw,thomas,jyeo}@cs.umd.edu

Introduction

The VQBD project addresses the following problem: What is the best way to *explore* an XML document of *unknown structure and content*? We focus on XML documents that are too large to browse in their entirety, even with the assistance of pretty-printing software (e.g., multi-megabyte or larger XML documents). In this context, we use the term *data exploration* to refer to the process by which a user gathers the information needed to use the data for a specific purpose (e.g., generating a report, writing queries, building user interfaces, writing applications). In a relational or object database, the schema (e.g., table definitions, class definitions, integrity constraints, and stored procedures) provides some of the information necessary for writing queries and applications. However, the schema is rarely sufficient for these tasks. Typically, one must probe and browse the database to discover data coverage, typical and exceptional values, and other information required to gain a better understanding of the database. In an XML environment, the need for such data exploration is much greater because it is quite likely that the XML data of interest is not accompanied by a schema. Indeed, much XML data is *semistructured*, meaning its structure is irregular, incomplete, and frequently changing. The rapid adoption of XML as a data exchange standard makes this *semistructured data exploration* problem increasingly important.

The VQBD system allows the structured exploration of arbitrary XML data. Although it focuses on XML data, an ancillary benefit is that VQBD is also a convenient tool for exploring relational or object databases (using trivial a mapping of relations and classes to XML elements). It's key features are summarized below:

- The subtasks of data exploration, viz., visualization, querying, and browsing, are complementary. For example, when a visualization module (e.g., one that plots cities on a map) has been applied to some query results (e.g., cities with no Starbucks stores), it is possible to refine the query through the visualization interface (e.g., by clicking points on the map). VQBD has a modular design with *plug-in* APIs that allow easy incorporation of additional visualization and querying modules.

- The *level of detail* presented to the user scales smoothly over a wide range. More precisely, the system is able to convey a useful summary of the data in a user-specified number of (graphical) objects. For example, if the display (or user) can only accommodate 20 objects, the system displays 20 objects that effectively summarize the database of, say, 50 thousand objects. These summary objects are only rarely simple reflections of database objects. At a low resolution, VQBD uses summary objects from graphical schemas similar to Data Guides in *Lore* [MAG$^+$97].

[*]We pronounce VQBD as "vee cubed." The VQBD team would like to thank Bongwon Suh for work on an early prototype. This work was supported by NSF grants in the CAREER (IIS-9984296) and ITR (IIS-0081860) programs.

- There is *no required structure*. The system provides acceptable results for any well-formed XML document. While it is reasonable to expect usability and performance to deteriorate as structure weakens, such deterioration is graceful, not catastrophic. As in semistructured databases, structure is descriptive, not prescriptive.
- Any available *explicit structure* is effectively used. For example, if the XML document is accompanied by DTD or RDF definitions to which it conforms, these definitions are used to provide an appropriate structured browsing interface.
- *Implicit structure* is detected and used. In addition to the explicit structure described by DTD, RDF, and similar methods, a given instance of XML data is likely to contain additional (implicit) structure and patterns. For example, although the DTD governing an XML document may permit address elements that are either strings (#PCDATA) or structured (line1, line2, city, state, zip), all addresses in the current instance may be in the latter format. This fact is used to simplify browsing and printing by always presenting addresses in the structured form. Further, the query-by-example interface is modified to signal that accurate searches of the form `address.zip = 12345` are possible since there is no danger of the ZIP code matching, say, a street number (as would be the case if searches were performed on string-valued address elements).
- It is easy to impose and use structure at run time. For example, an XML document may contain string-valued `name` elements. A user may notice (or know, from out-of-band sources) that all the name strings have a specific format (e.g., last name, first name, initials). The system permits such *run-time structure* to be specified at any time during the exploration process and, once specified, this structure is used in a manner analogous to implicit and explicit structure.

In addition to the above, the VQBD system embodies the *Perl* tenet of making simple tasks simple without making the more difficult tasks impossible. The system is designed for *exploration*, meaning the value it provides to a user scales well (both up and down) with user effort.

Many of VQBD's features have been separately considered by earlier systems. For example, Lore [MAG$^+$97] is built on the no-required-structure tenet, *Jazz* [BGM00] permits smooth zooming into graphical details, while *NoDoSE* [Ade98] facilitates run-time structure definition.) However, we believe VQBD is the first system to integrate and apply these ideas for the purpose of data exploration in XML. For example, VQBD combines the graphical zooming and panning provided by Jazz with a unique method for logical zooming (multiresolution data summarizing) and panning. Although the project is at an early stage, we have a fully functioning prototype that we believe is already the best way to explore XML data about which little or nothing is known. We now present the VQBD system in more detail, following it with a description of a sample demonstration session.

The VQBD System

Although VQBD is extensible and incorporates many advanced features for the power user, it is designed to be accessible to a casual user. To this end, the basic modes of interacting with the system are very simple. At all times, the VQBD display consists of a single window with a graphical representation of the XML data. Although, as we shall see below, this representation may be the result of some complex operations, the user interface is always the same: There are nodes (boxes) representing data elements (often summarized) and arcs (lines) representing relationships among them. There are no tool-bars, scroll-bars, sliders, or other widgets. We believe this simplicity is key to usability by a casual user. The basic modes of controlling VQBD are also simple and unchanging. The first three are meant for the novice user, while the next two are for users who have gained more experience with the system.

Panning The displayed objects can be moved in any direction relative to the canvas by a dragging motion with the left button of the mouse.

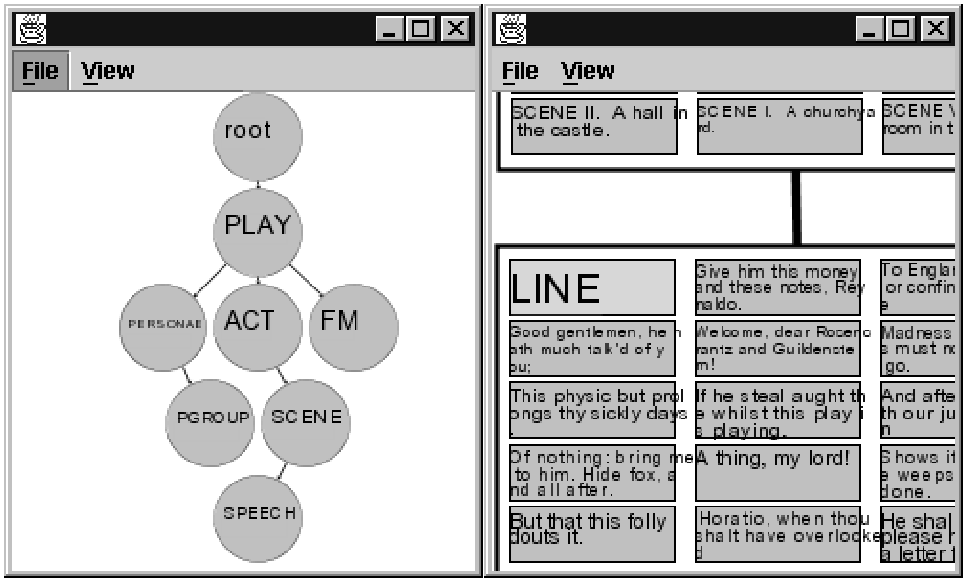
Zooming The display may be zoomed in (or out) by a right- (respectively, left-) dragging motion with the right mouse button. VQBD uses the position of the pointer to determine the type of zooming. If the pointer is outside all graphical object then the result is simple graphical zooming (e.g., larger objects, bigger fonts). If the pointer is inside a graphical object then the data resolution of that object, and any others of a similar type, is increased. For example, consider the screenshot in Figure . The lower part represents *speech* and *line* objects and includes sample values from the input document. Zooming in with the pointer inside the larger box (representing the *collection* of line objects) results in the display of a larger number of sample speech objects. Zooming in with the pointer inside one of the smaller boxes representing an *individual* line object displays that object in more detail (more text). In the case of other visualization modules (e.g., histograms), zooming results in actions appropriate to that module (e.g., histogram refinement).

Link Navigation Clicking on a link causes the display to recenter itself around the target of the link at an appropriate zoom level. Following the design method of the *Jazz* toolkit, such link navigation is not instantaneous; instead it occurs at a speed that allows the viewer to discern the relative positions of the referencing and referenced objects. In addition to selecting an appropriate graphical zoom level, VQBD automatically picks a suitable *logical zoom level.* For example, a collection of numbers that is too large to display in its entirety is often presented as a histogram.

View Change While VQBD automatically selects an appropriate method for visualizing data at the available resolution, the user may override this selection a pop-up menu bound to the middle mouse button. For example, a user interested in the highest values in a collection of numbers may force VQBD to change the view from histogram to sorted list.

Querying The XML document may be queried using a query-by-example interface. This interface permits users to specify selection conditions as annotations on displayed objects. In addition, the user may mark objects as *distinguished objects* for use in queries. Intuitively, these objects can be used as the starting points for query-based exploration. VQBD has built-in query modules for regular expressions and XPath. Additional query modules can be easily added using the plug-in interface.

VQBD is implemented using Java (JDK 1.2.2). For its display routines, VQBD uses the Java *Swing* graphics library and the Jazz zoomable user interface library, version 1.0 [BGM00]. It uses IBM's *XML4J* parser, version 3.0.1, to extract and manipulate a DOM representation of XML. VQBD has a modular design and runs on any platform that supports the Java run-time environment. We have tested it on Solaris, Linux, and NT, with different Java runtime environments, and different parsers (e.g., *Jaxp*, version 1.1).


File
View
root
PLAY
PERSONAE
ACT
FM
PGROUP
SCENE
SPEECH
SCENE II. A hall in the castle.
SCENE I. A churchyard.
LINE
Give him this money and these notes, Reynaldo.
Good gentlemen, he hath much talk'd of you;
Welcome, dear Rosencrantz and Guildenstern!
This physic but prolongs thy sickly days
If he steal aught the whilst this play is playing,
Of nothing: bring me to him. Hide fox, and all after.
A thing, my lord!
But that this folly douts it.
Horatio, when thou shalt have overlooked

(a) Zoomed out—structural summary (b) Zoomed in—instances

Figure 1: Two screenshots of VQBD in action

Sample Demonstration Session

We outline here some of the key points in our demonstration of VQBD. For concreteness, we describe a session based using Jon Bosak's XML rendition of Shakespeare's *A Midsummer Night's Dream*, available at `http://www.ibiblio.org/xml/examples/shakespeare/`. The system parses the file and graphically and presents a summary of the file's implicit structure with objects representing the play, acts, scenes, and lines. This structural summary is the first view presented by VQBD. A screenshot appears as Figure . Note that the screenshots in Figure 1 are based on a rather small VQBD display (approximately 350x350 pixels). While we picked this size primarily to fit the space constraints of this report, it also illustrates how VQBD's zooming interface allows it to function effectively at this size.

We then zoom in on the speech object using a dragging motion with the right mouse button. Initially, the zooming results in standard graphical results (larger objects, higher resolution text, etc.). However, as soon as the object becomes large enough to display graphical elements within it, the graphical zooming is accompanied by a logical zooming: a few sample elements are displayed. VQBD displays randomly sampled elements, with the number of displayed elements increasing as the available space increases as a result of the zooming in operation. Figure is a screenshot at this stage of exploration. In addition to details of the speech and line elements, details of scene elements (appearing above the speech elements in this figure as in Figure) are partial visible, providing a useful context. These figures do not convey the colors used by VQBD for indicating many relationships, including grouping elements based on parents (enclosing elements). When a sample element is displayed in this manner, VQBD reads its attributes and sub-elements to pick a short string that distinguishes the element from others with the same tag. This string is displayed within the object representing the element on screen. In our example, VQBD uses the scene titles to identify scene elements on screen. At this stage, the user also has the option of single-clicking on any of the displayed objects, causing VQBD to display all details of the selected object. For example, clicking on the scene object labeled *A hall in the castle* results in displaying the scene in greater detail (as much as will fit in the VQBD window). Note that this clicking action is simply an accelerated form of zooming; the same result could be achieved by zooming in to the scene object. Subelements of the scene element are displayed as active links that can be activated in order to smoothly transport the display to the referenced object. This link-based navigation can be freely interleaved with zooming. Zooming out, we observe VQBD retracing its steps, displaying data in progressively less detail until we are back at the original structural summary view.

We demonstrate the query features by selecting the scene object and searching for sub-objects matching the string *Lysander*. VQBD executes the query and highlights objects in the query result. In our sample data, the query string matches elements of different types (two *persona* elements, one *stagedir* element, and several *speaker* and *line* elements). Since the current resolution is insufficient to display individual objects, only the structural summary objects corresponding to the individual objects are highlighted. To view the query results in detail, we zoom in as before. Unlike the earlier zooming action, which displayed a random sample of all elements corresponding to the summary object, VQBD now displays a sample chosen only from the elements in the query result. When all elements in the query result have been displayed, further zooming results in a random selection from the remaining elements (as before). (Colors are used to distinguish the elements in the query result from the rest of the elements.) This exploration of query results may be interleaved with zooming, panning, query refinement, and other VQBD operations.

The above description is a simple run of VQBD that displays only some of its features. As described above, our fully functional implementation supports many other features that will also be demonstrated. For example, we will demonstrate visualization modules such as histograms (for numeric data) and pie charts (to display the relative number of subelements by tag). We will also demonstrate VQBD on other XML documents culled from diverse sources (e.g., XML-ized Web pages, an XML document describing an ontology, and search engine results). In addition, since our system works with any XML document, we will invite participants to try VQBD on their own documents.

Our VQBD demonstration will introduce the audience to the interesting and important problem of semistructured data exploration and should help spur further work in this area. The VQBD system is a unique synthesis of diverse techniques from the data management and information visualization fields. Although it is in an early stage, we believe VQBD is already the best tool for exploring XML data of unknown structure and content.

References

[Ade98] B. Adelberg. NoDoSE—a tool for semi-automatically extracting semi-structured data from text. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, June 1998.

[BGM00] B.B. Bederson, L. Good, and J. Meyer. Jazz: An extensible zoomable user interface graphics toolkit in Java. In *Proceedings of User Interface and Software Technology*, 2000. To appear.

[MAG$^+$97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.