BBQ: A Visual Interface for Integrated Browsing and Querying of XML

Kevin D. Munroe, Yannis Papakonstantinou

University of California, San Diego, La Jolla, CA 92093

{kmunroe, yannis}@cs.ucsd.edu

Abstract

In this paper we present BBQ (*B*lended *B*rowsing and *Q*uerying), a graphic user interface for seamlessly browsing and querying XML data sources. BBQ displays the structure of multiple data sources using a paradigm that resembles drilling-down in Windows' directory structures. BBQ allows queries incorporating one or more of the sources. Queries are constructed in a query-by-example (QBE) manner, where DTDs play the role of schema. The queries are arbitrary conjunctive queries with GROUPBY, and their results can be subsequently used and refined. To support query refinement, BBQ introduces *virtual result views*: standalone virtual data sources that (i) are constructed by user queries, from elements in other data sources, and (ii) can be used in subsequent queries as first-class data sources themselves. Furthermore, BBQ allows users to query data sources with loose or incomplete schema, and can augment such schema with a DTD inference mechanism.

1. Introduction

As the World-Wide Web comes to be viewed as a large semistructured database with XML [XML98] as its model, issues related to querying semistructured data in general, and XML in particular, become more important. Unlike relational or object-oriented databases, semistructured data sources have content structure that is too irregular to easily map to a rigid schema. Recent research in this area has focused on query languages [BDS95, BDH+96, AQM+97], semistructured data extraction [HGC+97] and systems for integrating heterogeneous sources [CMH+94, MAG+97, BLP+98]. Research has largely ignored the issue of user interfaces for browsing and querying semistructured data; we believe this issue will be of great importance in the near future as XML becomes more widely by the general public.

The MIX (*M*ediation of *I*nformation using *X*ML) project [BLP+98] exports to clients integrated views of autonomous XML data sources, using a well-known mediator architecture [Wie92]. The focus of this paper is the MIX client - a graphical user interface called *BBQ* (*B*lended *B*rowsing and *Q*uerying). BBQ facilitates intuitive querying and browsing of one or more XML data sources, seamless iterative query refinement, and structure discovery. Data sources are displayed in a multi-document interface (MDI) layout- each source is assigned a window, with its data and schema shown side-by-side within the frame. Both the data and the schema are displayed as directory-like tree structures, which users can navigate and place conditions on.

BBQ combines the search paradigm used in relational databases with Web searching. As with relational database searches, querying in BBQ is schema-driven (using XML DTDs); but like web searches we assume that the user is not

in advance sure of what is the focused query result he is interested in. So the system emphasizes navigation of schemas and data and the query results can be iteratively refined.

We support query refinement by having query results be sources used in subsequent queries. Users can construct a query result document (essentially a virtual view) and that document becomes a first-class data source within BBQ, meaning it can be browsed, queried, or used to construct another query result document. We facilitate the perception of a query result as a source by automatically inferring its DTD from the query statement and the source DTDs using MIX's view inference module [PV99]. Furthermore, once the data for the query result are obtained the structure discovery module figures out the structure of elements that were loosely specified (using the keyword ANY) in the source DTDs and displays it accordingly.

BBQ is built using the MIX mediator's DOM client API, which provides an efficient platform for virtual views. In particular, the MIX mediator's *lazy* implementation of DOM, called DOM-VXD [LPV99], does not actually retrieve source objects until they are absolutely necessary. Hence it is an ideal framework for browsing large query results where the user may eventually browse only a small part of the query result and then he refines the query.

The intended audience of the BBQ interface is persons who understand the nested structure of XML data and have both the need and the expertise to use database query language concepts such as joins and aggregates. A typical user would be a webmaster or a database administrator/programmer. However, BBQ has a programming interface that allows for the creation of simpler user interfaces, if one would be so motivated as to design one for less sophisticated users. This is discussed further in Section 4.3.

1.1. Related Work

There are very few visual interfaces for querying and browsing semistructured data, and fewer still for XML. The bulk of the effort on semistructured data is focused on query language design, processing, schema, view inference, etc.; interfaces appear to be tacked on merely as test drivers. It is therefore no surprise that BBQ was most strongly influenced by a visual interface designed *not* for semistructured data, but for object databases- IBM's PESTO [CHM+96]. PESTO's design introduced the notion of fusing query formulation and result browsing into one seamless paradigm, termed Query-In-Place. However, PESTO's use of temporary *synchronizers* for viewing joins across multiple databases breaks down the seamlessness; users can browse tuples in the synchronizer, but cannot built queries on top of them. Also, as an object database interface, PESTO is not equipped to display irregular schema common to semistructured data. [GGK99] is similar to BBQ in that schema is presented in a graphical manner, and a special mechanism (a "copy and drop" feature) allows users to use sample data from results as constraints in subsequent queries.

Visual interfaces that *can* display irregular schema, such as those found in EquiX [CKK+99] and Lore's DataGuides [GW97, GW98], do not support queries across multiple DTDs. Further, in DataGuides the tree structures do not accurately render the data's structure; conjunction, disjunction, and repetition ('?', '*', etc) are not shown. Omitting structural clues can lead users to create unsatisfiable queries, for example if the user asks for a sequence consisting of *condition1(a)* and *condition2(a),* when in reality only one a exists (a is not adorned with '*' or '+'). One system that accurately displays DTDs (wildcards and all) is the recent IBM Alphaworks' VisualDTD [IBM99],

although VisualDTD functions as a DTD composition tool only. The Araneus project [MAM+98] allows for querying of HTML pages using POLYPHEMUS, a visual interface that shows connected web pages as a large diagram.

1.2. Paper Outline

Section 2 describes the basic look and feel of BBQ. Section 3 covers the BBQ query cycle, and issues related to query formulation and refinement: XMAS [LPV+99] query construction, creation of virtual views, and incremental refinement of DTDs. Section 4 covers BBQ ''accessories" and the application programming interface. Section 5 presents BBQ's current implementation status and discusses future work.

2. The BBQ Interface

In this section we introduce the reader to BBQ's representation of XML data, and its uniform interface for structural navigation of both data and schema. First, we describe some sample DTDs that will be used in examples throughout the paper.

```
<!DOCTYPE CSEStudents [                          <!DOCTYPE Interns [
<!ELEMENT CSEStudents   (CSEStudent)*>           <!ELEMENT Interns (Intern)* >
<!ELEMENT CSEStudent   (name, advisor?, degree)> <!ELEMENT Intern (name, supervisor,
<!ELEMENT name          (#PCDATA)>               sponsor) >
<!ELEMENT advisor       (#PCDATA)>               <!ELEMENT name       (#PCDATA) >
<!ELEMENT degree        (#PCDATA)>               <!ELEMENT supervisor (#PCDATA) >
]>                                               <!ELEMENT sponsor (CAIDA | NPACI | UCSD)>
                                                 <!ELEMENT CAIDA    (ANY)>
                                                 <!ELEMENT NPACI    (ANY)>
                                                 <!ELEMENT UCSD     (ANY)>
                                                 ]>
                                                              (b)
                        (a)
```

```
<!DOCTYPE ResHalls [
<!ELEMENT ResHalls (ResHall | RitzyResHall )+>
<!ELEMENT ResHall  (hallName, apartments)>
<!ELEMENT RitzyResHall (hallName, apartments)>
<!ELEMENT hallName  (#PCDATA)>
<!ELEMENT apartments(apartment)+>
<!ELEMENT apartment (floor, number, tenant)>
<!ELEMENT floor     (#PCDATA)>
<!ELEMENT number    (#PCDATA)>
<!ELEMENT tenant    (#PCDATA)>
]>
                        (c)
```

Figure 1: A set of base DTDs

.

2.1 Base DTDs

BBQ is XML-driven, i.e. schema and data instances are represented by XML DTDs and XML documents respectively. Figure 1 shows a set of DTDs exported by the MIX mediator. We refer to these DTDs as *base DTDs*. A base DTD may

be the integrated view of multiple data sources, but BBQ need not be aware of this; BBQ thinks of each DTD as emanating from a single XML data source.

Let us assume that the CSEStudents DTD source shown in Figure 1(a) is a database of graduate students in UCSD's Computer Science program, Interns in Figure 1(b) is a database of summer interns at SDSC, and ResHalls in Figure 1(c) is a database of UCSD's Residence Halls tenants. The CSEStudents relation contains zero or more CSEStudent elements. Each CSEStudent has a name, a degree, and perhaps an advisor. The Interns relation contains zero or more Intern elements. Each Intern has a name, a supervisor, and a sponsor. Note that the intern's sponsors, CAIDA, NPACI and UCSD, are described with the special XML token 'ANY', meaning they can be any XML subtree. The ResHalls relation contains one or more of either ResHall or RitzyResHall elements. Each ResHall/RitzyResHall contains a name, address, and list of one or more apartment elements. The apartment consists of the floor number (floor), apartment number (number) and the tenant's name (tenant). Finally, each of these data sources has an associated set of XML document instances.
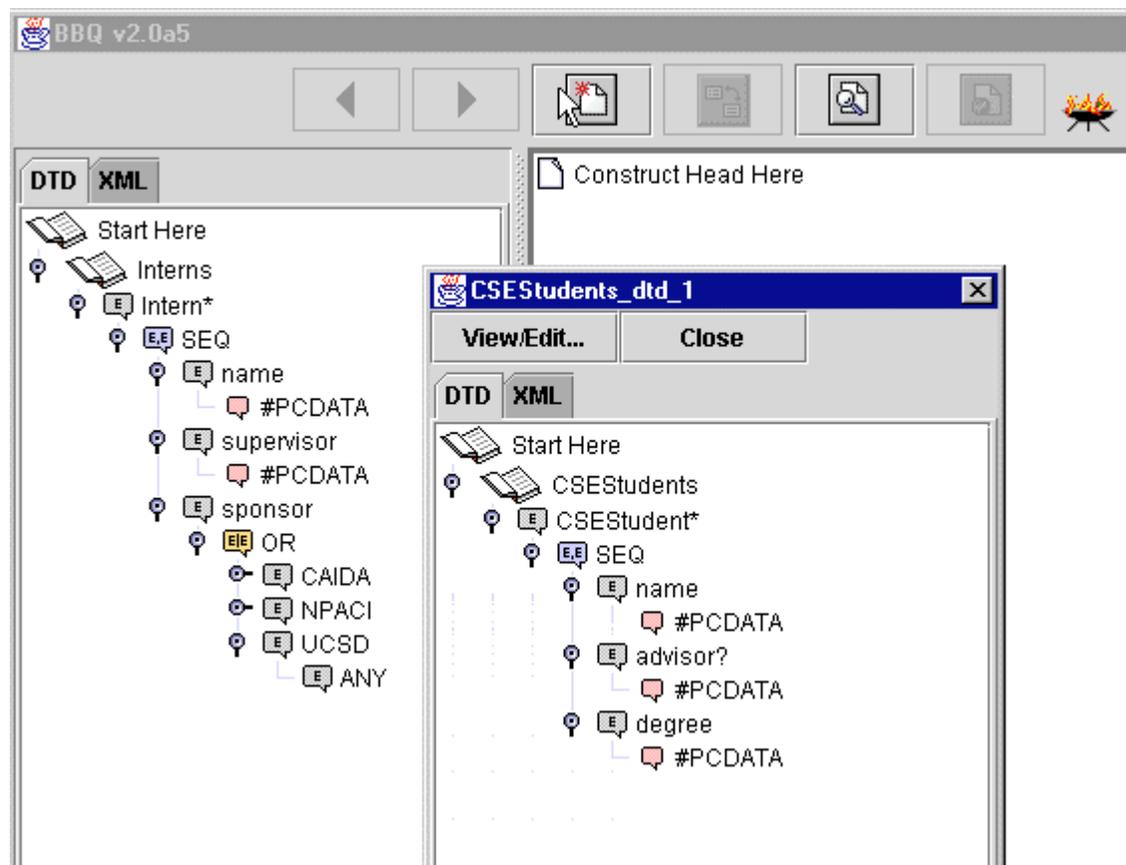
2.2 The Interface

Figure 2: The BBQ Interface

The BBQ interface consists of one main window and zero or more floating windows. The main window contains a toolbar, a split pane, and a message console, while the floating windows contain a toolbar and split pane only. BBQ is a

standalone Java application; at the time of this writing, a well-known Java Swing bug prevents BBQ from functioning properly as an applet[1]. Eventually, BBQ will be available as both.

We begin our example, by selecting and browsing two DTDs exported by the mediator (Figure 2). This is the beginning of a *query session* and the beginning of the first *query cycle*; Section 3 discusses query cycles in detail.

The DTDs are represented as trees in the obvious hierarchical manner: an element name is a parent node, and that element's sub-elements (defined in its **contentspec**) are its children. BBQ features special tree nodes to represent XML DTD's structural operators such as the **choice** and the **seq**(uence)[XML98]. For example, Figure 2 shows the CSEStudent element with one child node labeled SEQ, with CSEStudent's **contentspec** as SEQ's children, and advisor labeled with a question mark (?). This is the visual equivalent of :

<!ELEMENT CSEStudent (name, advisor?, degree)>.

These special tree nodes give the user a more accurate view of the DTD's structure than other semistructured-data viewing systems, and they also facilitate more complex queries. For example, a default order constraint is introduced, namely the one that corresponds to the order in which elements are listed on the screen. The users can create queries where sibling order is irrelevant, by right-clicking on the SEQ node and de-selecting "Constrain Order". In the display SEQ becomes AND, and internally the order constraint will not be applied when the query is generated.

Figure 2. illustrates that BBQ can simultaneously display multiple DTDs. This feature may seem simple and obvious, yet it is powerful and surprisingly unique among semistructured-data display tools we have came across. Its power comes from the type of queries it allows- joins and functions across databases. In the main BBQ window, at most one DTD can be displayed, in the left panel. Each subsequently requested DTD is displayed in a floating window, and has a unique string appended to its name in the event that the same DTD is opened in multiple windows. The DTDs can be browsed and queried independently, or combined in queries.

At the top left of the main BBQ window are BACK and FORWARD arrows. These buttons let the user go back to previous query cycles. To the right of the FORWARD arrow is the VIEW NEW DATA SOURCE button, which brings up the list of exported DTDs. Next is the VIEW/EDIT JOINS button, which shows a dialog box containing an editable list of joins set so far in this query cycle. To the right of the VIEW/EDIT JOINS button is the VIEW SOURCE button, which brings up a dialog box containing a text version of the main window's DTD. Finally, to the right is the EXECUTE QUERY button, which triggers BBQ to complete the query cycle by (among other tasks) composing an XMAS query, sending it to the MIX mediator, and displaying the results.

The right-hand panel of the main BBQ window is where XMAS queries with conditions on multiple DTDs is constructed. The node labeled 'Construct Head Here' is the "root" upon which the query result document is built. Section 3.2 describes the role of this panel in the query construction process.

2.3 Browsing DTDs And XML

BBQ displays each requested data source in a tabbed pane set- one tabbed pane shows the DTD, the other shows a set of corresponding XML instance data, represented as a directory tree (Figure 3). We mentioned earlier that query result instances are materialized on demand via DOM-VXD. It turns out that base DTD instances are retrieved

---

[1] 09/10/99, see http://www-cse.ucsd.edu/users/kmunroe/MIX/BBQ.html for updates.

from the mediator using the same mechanism. The XML data is materialized on demand from the source, in customizable increments. The buttons labeled **next** and **previous** in the XML panel retrieve the next and previous *n* instances, respectively. We describe an *instance* as a child subtree of the XML document root element. As is obvious from the figures the presentation and browsing of both schema and data is done using the well-known paradigm of navigating into a directory structure.
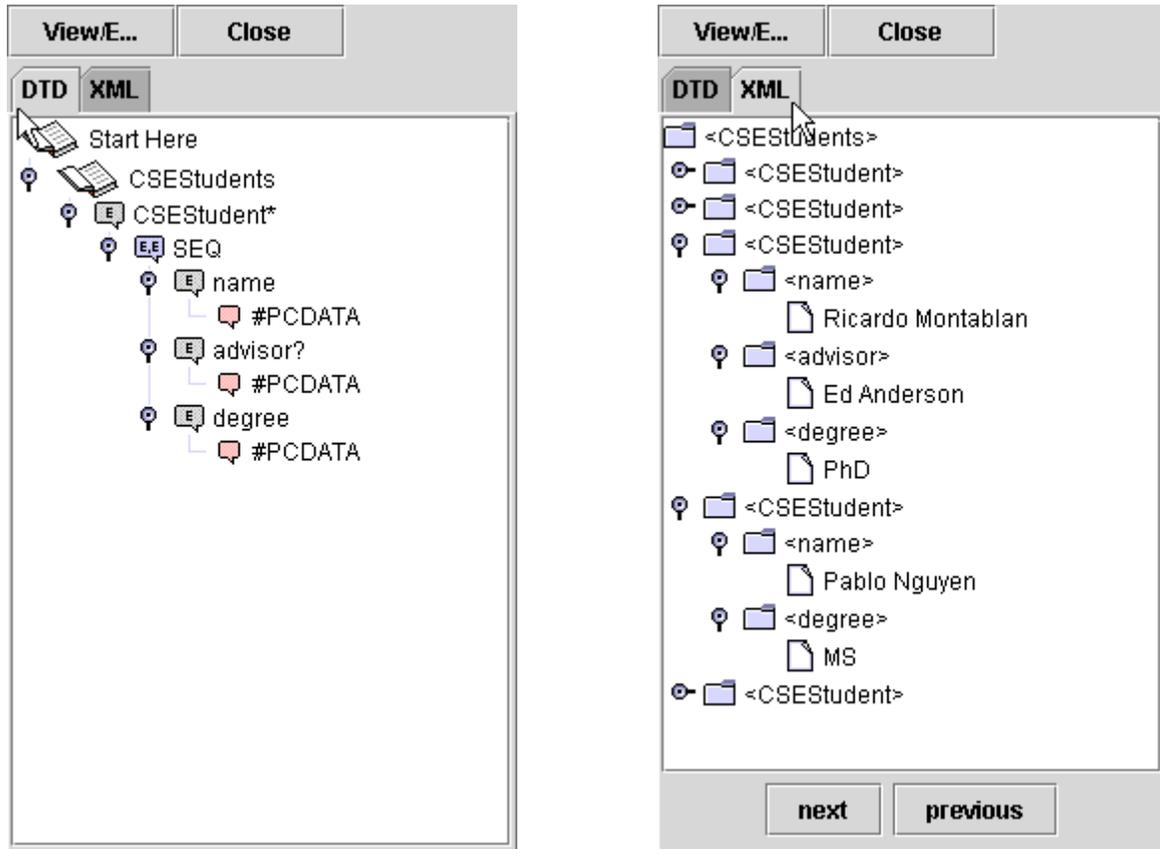
(a)   (b)

Figure 3: CSEStudents DTD (a), and corresponding XML document (b)

3. A BBQ Query Cycle

In this section we describe BBQ's intuitive query formulation, its XMAS query generation, and its unique query refinement capabilities. We first define *query session* and *query cycle*, then step through an example, highlighting BBQ features along the way.

A *query session* is the set of events that occur while BBQ is connected to the mediator. Each query session consists of one or more *query cycles*. A query cycle is the set of events that starts with the user constructing a query, and ends with the user browsing the query result. The basic BBQ query cycle takes place in four steps. First, constraints are set on the data sources. Second, a tree representing the query result schema is created by dragging and dropping elements. Third, the XMAS query is generated and submitted to the mediator. Fourth, a DTD is generated for the query

result and the query result schema and data are displayed. (Recall, generating a schema for the query result is done using just the query statement and the source DTDs, as described in [PV99]).. XMAS queries are of the form

CONSTRUCT *head* WHERE *body*

The *body* describes the constraints to be set on data to be extracted, and the *head* describes the data to be displayed. From the above discussion it should be clear that step one of the query cycle corresponds to constructing the XMAS body, and step two to the head. The following two sections show by example how to set constraints and construct the head respectively.
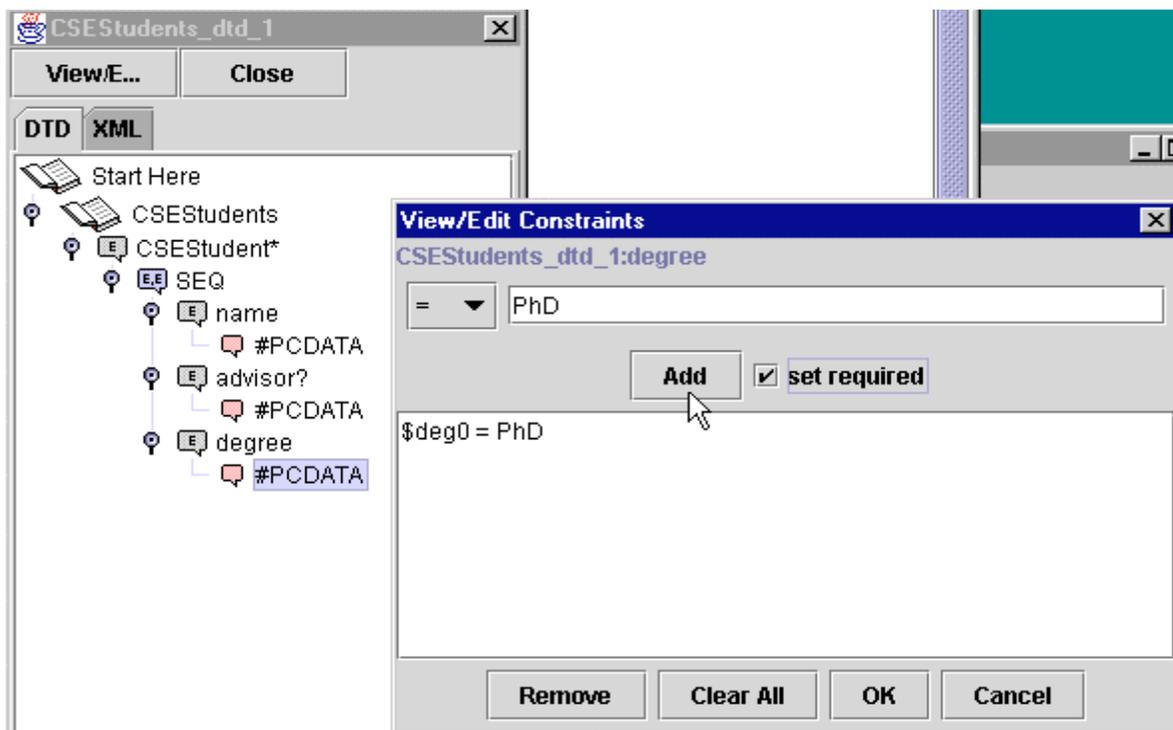

Figure 4: Setting constraint on the `contentspec` of `degree`

3.1. Setting Constraints, Joins, and Filtering

Constraints can be set on the leaf nodes of the DTD tree or XML tree. Constraints cannot be set on nonleaf nodes (**contentspec** that are not terminals like **#PCDATA** or **EMPTY**) only because XMAS does not support element constraints that are more complex than the default existential one. The operators are a basic set of comparators ('=','<=', '>=', '<', '>', 'substr'); in the future we anticipate mediators exporting a list of special operators they support.

As an example of constraint setting, Figure 4 shows the user restricting `CSEStudents` to doctoral-track students only. The user right-clicks the `degree` element and selects "View/Edit Constraint..." from the popup menu. This action brings up the "View/Edit Constraint" dialog box, where "=" is selected as the operator, and "PhD" is typed in as the operand. At this point, the user clicks "OK", and `degree` is visibly marked with the constraint (not shown). No actual data manipulation takes place until the query request is sent to the mediator and a result is returned.

Joins can take place within a data source or across data sources. Creating a join in BBQ is as simple as selecting one leaf element, and dragging and dropping it onto another leaf element. Continuing our example, suppose the user is interested in doctoral-track CSEStudents who are also interns, and whose advisor is also their internship supervisor. Figure 5 shows a join across CSEStudents and Interns, where CSEStudents.advisor is being bound to Interns.supervisor. CSEStudents.name is bound to Interns.name in the same manner.

Taking a page out of PESTO's playbook, BBQ provides a mechanism similar to Query-In-Place, which we call *filtering*. Filtering implicitly performs the last three steps of the query cycle; the filter is immediately evaluated and results are made available without explicit construction of the head. In particular, filtering takes place as follows. While browsing the XML or DTD tabbed pane, the user may decide to constrain some elements of a collection, and view only those elements that satisfy the constraint. The constraint is set just as shown in Figure 4, but after dismissing the constraints dialog with "OK", this time the user hits the EXECUTE QUERY button. The query result DTD is the DTD of the constrained selected elements and is computed by the view inference module [PV99] that is able to change the source DTD structure in order to account for the "refinement" induced by the conditions. For example, if the user places a condition on an optional subelement *s* of the selected element *x* the query result DTD indicates that now *s* is actually a required field of *x* (since if *s* were absent *x* would not qualify for the answer.) The XML tabbed pane now shows only
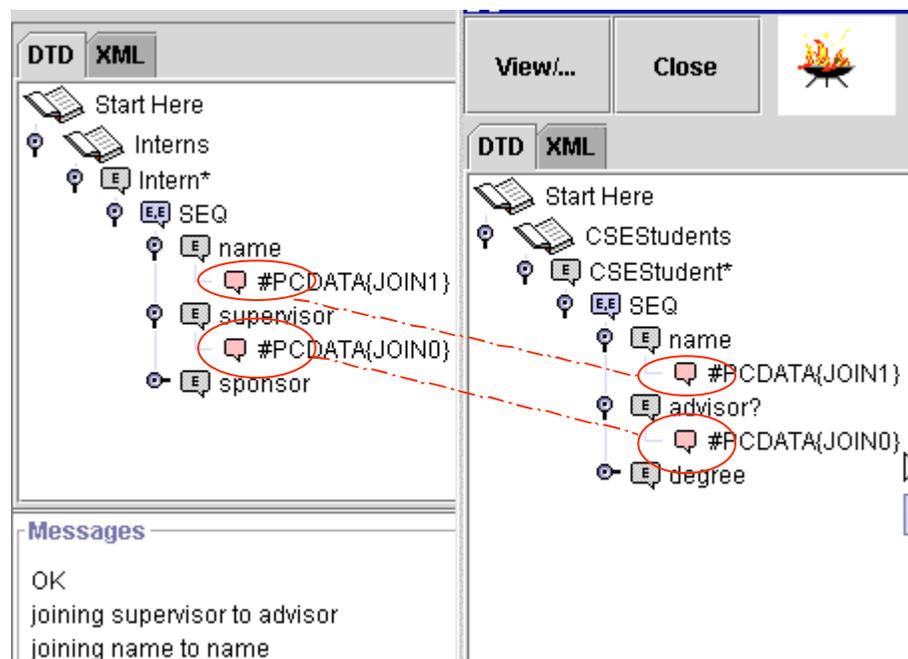

Figure 5: Setting a join across data sources. After supervisor's contentspec is dragged and dropped onto advisor's **contentspec**, both are marked with a join number {JOIN0}. Likewise for Intern.name and CSEStudents.name.

XML data that satisfy the constraint. In general, if the head panel is empty and the user selects EXECUTE QUERY, a filter is executed on the body of the panel where the button was pressed.
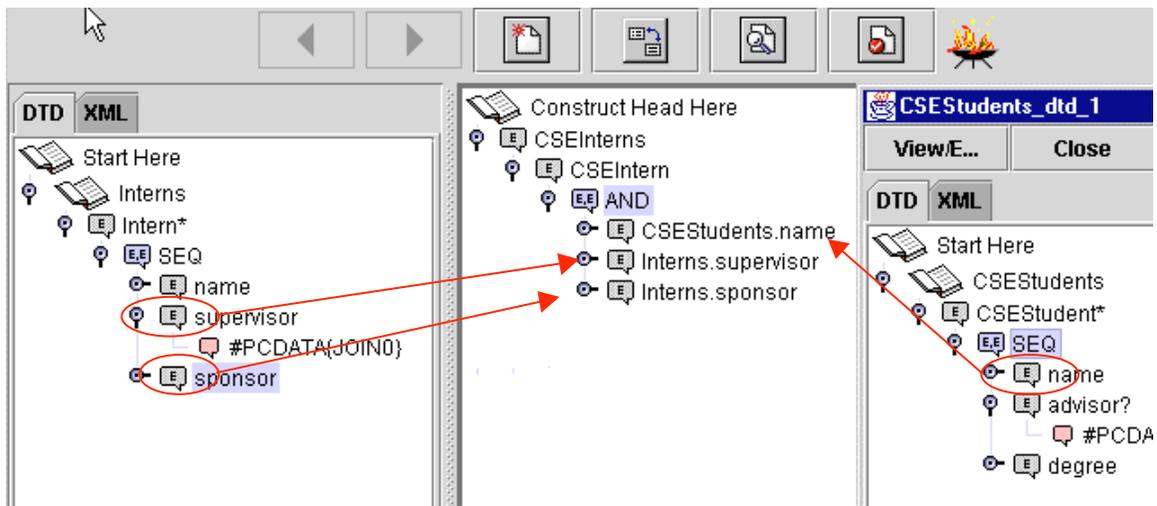
3.2 Constructing The Head

Figure 6: Constructing the head with elements from both sources, using drag-and-drop

Once the constraints are set, the next step in the query cycle is to construct a tree that the answer document(s) must conform to, called the head or *query result tree*. The right panel of BBQ's main window is where the head is built. The head is composed of elements (and their sub-trees) dragged from source DTDs, and tags created on the spot with the "Create New Child" popup menu item. The query result tree must be a tree, and BBQ enforces this by allowing only a single root element. Figure 6 shows a query result tree that contains a newly-made root element CSEInterns, a newly-made child CSEIntern, and three elements- name dragged from CSEStudents, supervisor and sponsor dragged from Interns.

The "{GRP+}" string marking the CSEIntern element in Figure 7(b) indicates an *XMAS GROUP-BY label*. A GROUP-BY label is roughly equivalent to GROUP-BY semantics of OQL; in our example, it indicates that there will be one CSEIntern element for each unique Interns.name element. To set GROUP-BY labels, the user right-clicks on an element in the head, and selects the "Group By:Other…" popup menu item. A dialog appears, and until the dialog is dismissed, any element the user selects is added to the list of GROUP-BY elements. Figure 7(a) shows Interns.name being selected as a group-by element for CSEIntern.
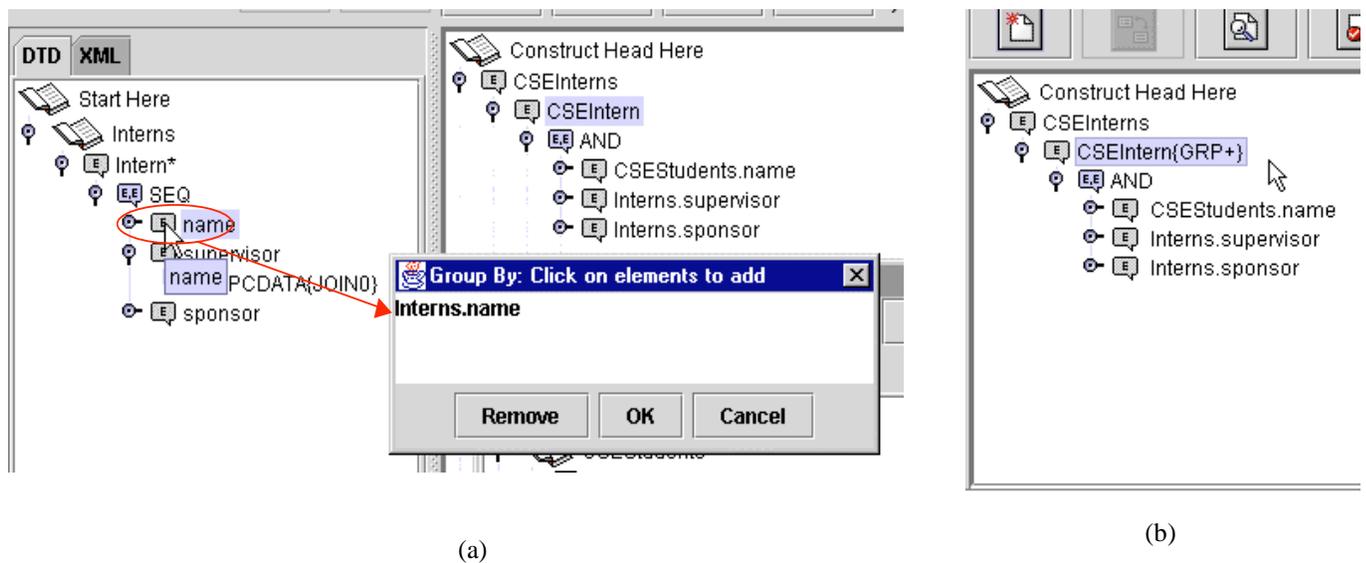
(a)

(b)

Figure 7: Interns.name is selected as the GROUP-BY element (a), and CSEIntern is marked (b)

3.3 Executing the Query

The query we have constructed is now equivalent to *"SELECT CSEStudent.name, Interns.supervisor, Interns.sponsor WHERE CSEStudent.degree = 'Ph.D' and CSEStudent.advisor = Interns.supervisor; Group by CSEStudents.name, within the element CSEIntern"*. (This query notation is given just for understanding purposes. Figure 8 shows the actual query.) Once the user clicks the "Execute Query" button, we enter the third step of the query cycle. BBQ converts the visual layout into XMAS query language, contacts the MIX mediator and submits the query. Figure 8 shows the generated XMAS query. We do not get into the details of XMAS syntax and semantics. We merely note that the WHERE clause follows XML-QL's [XML-QL] syntax and semantics while the SELECT clause has replaced XML-QL's grouping by Skolem-id's with the more conventional grouping using group-by. Nevertheless, observe that this relatively complex query statement was created with a few simple GUI actions, and with no knowledge of the underlying query language.

The conversion step from GUI layout to XMAS query can be straightforward or quite complex. Constructing an XMAS query such as Figure 8 is relatively simple: assign variables to constrained elements and head elements, state head element variables in the CONSTRUCT clause, state the tree paths that lead to those elements in the WHERE clause, and finally state all constraints at the end of the WHERE clause. A more complex case is where a subtree is dragged to the head, and a node of that subtree is deleted. In this situation the tree path of the deleted node, and *all the the siblings of the deleted node* must be explicitly enumerated in the WHERE clause so that the query can manage to keep everything but the deleted element.

```
XMAS Query Preview                              ×

CONSTRUCT
 <CSEInterns>
  <CSEIntern> $nam4 </CSEIntern> { $var5 }
 </CSEInterns>

WHERE

 <CSEStudents>
  <CSEStudent>
     $nam4:<name> $nam2     </name>
     <advisor> $adv1     </advisor>
     <degree> $deg6     </degree>
  </CSEStudent>
 </CSEStudents>
IN  CSEStudents_dtd_1 AND

 <Interns>
  <Intern>
     $var5:<name> $nam3     </name>
     <supervisor> $sup0     </supervisor>
  </Intern>
 </Interns>

IN  Interns_dtd_0
AND $deg6 = PhD
AND $adv1 = $sup0
AND $nam2 = $nam3


         Preview    Execute    Cancel
```

Figure 8: BBQ generates this XMAS query corresponding to GUI object layout.

4. Miscellaneous Accessories

4.1 Vertical Wildcards

BBQ offers a unique feature for visually querying elements that can be reached by multiple paths: the vertical wildcard. It allows users to target elements without specifying their exact location in the DTD hierarchy. Suppose while browsing Figure 1(c), our user becomes interested in tenants living on the 3rd floor, regardless of living arrangements (ResHall or RitzyResHall). BBQ displays apartment as a subtree of both ResHall and RitzResHall; the naive solution would be to find all instances of apartment, and for each instance set the floor condition, and drag tenant to the head. For larger DTDs it may be too time-consuming to find all instances of an element, so BBQ offers the following solution: the user selects the ResHalls element, then selects "Vertical Wildcard" from the

popup menu. BBQ attaches a node labeled "ANY" to `ResHalls`; the user then drag and drops one of the `apartment` elements onto the "ANY" node. Finally, the user sets the 3rd floor condition on `apartment.floor`, and drags `apartment.tenant` to the head. Figure 9(a) shows the construction of the query, and Figure 9(b) shows the resulting XMAS query[2].
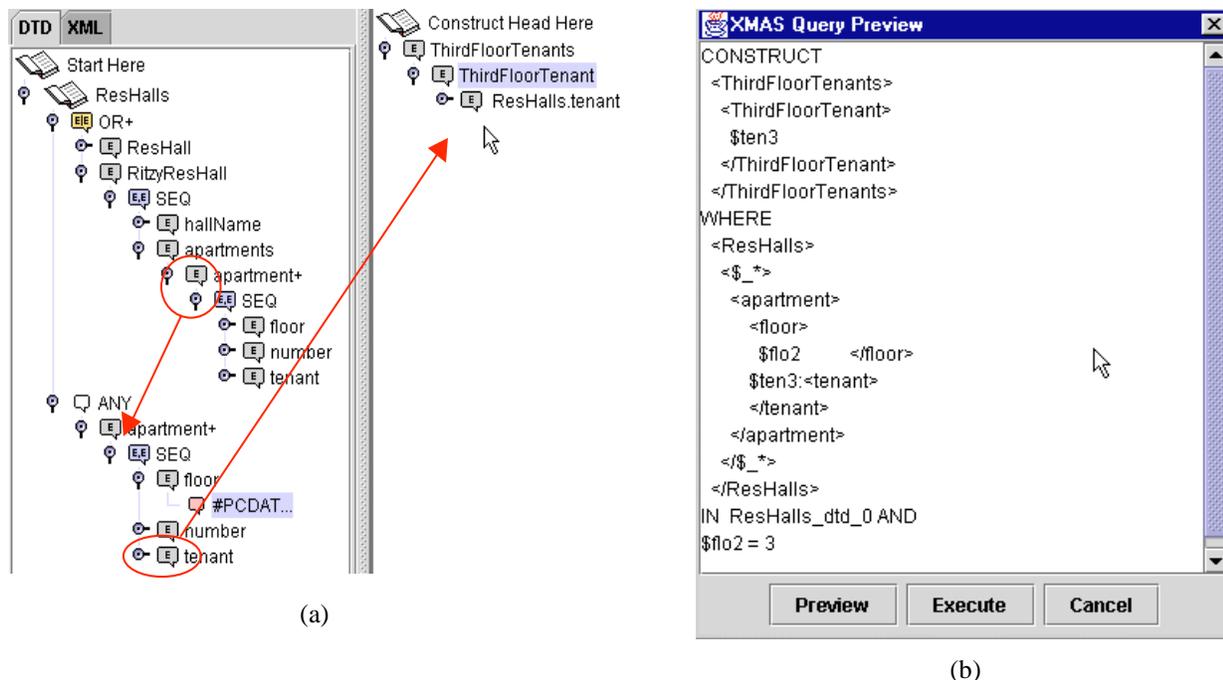

(a)

(b)

Figure 9: User adds "ANY", then drags one of the apartment elements to child position, sets condition on floor, drags tenant to head (a). The resulting XMAS query (b).

In general, users can drill down to any element in a DTD, attach the "ANY" node, then drag and drop any subtree from that DTD onto the "ANY" node. Once users set constraints on elements in the subtree, the vertical wildcard appears in the XMAS query; otherwise it is ignored. The vertical wildcard is a powerful construct that we have not observed in any other visual querying system for semistructured data to date.

4.2. Query History

BBQ allows the user to explore the last n query cycle transactions, where n is a configurable positive number. Each click of the BACK button takes the BBQ state to the end of the previous query cycle, just before EXECUTE QUERY or "Query" was clicked. All constraints set at that point are re-bound to their respective elements, so the user can remember how an answer document was obtained.

4.3 The BBQ Application Programming Interface

In some query domains, BBQ's current interface may not be the most suitable display paradigm; for example an XML-based Geographic Information System (GIS) database may prefer to display its data as a clickable map.

---

[2] In XMAS, <$_*> is equivalent to the XML "ANY" token.

Therefore, to make BBQ applicable in different environments, we have decoupled the user interface from the mechanisms which manage the data structures and query sessions. The underlying mechanisms have been bundled together in a package with a simple programming interface, called the BBQ Application Programming Interface (BBQ API)[3]. BBQ API aims to abstract away as much of BBQ's inner workings as possible, while allowing users to safely manipulate some of the core data structures. Specifically, BBQ API gives client applications read-only access to tree representations of all base DTDs, source DTDs, answer trees, and all their corresponding XML documents. The XML trees are represented by Java 2's `DefaultMutableTreeNode` class, while the DTD trees are represented by our own `BBQTreeNode` class whose methods mirror `DefaultMutableTreeNode`. By exposing the document data structures, we give client applications a great amount of flexibility; they are free to navigate about the trees and display their contents in any fashion. Also, by keeping the data structures read-only, we simplify our own internal book-keeping while preventing users from misaligning their view of the data source with the "real" view maintained by the mediator.

The core of BBQ API is in the `BBQSession` class. Within BBQSession are all the calls necessary to perform a query session; a typical client application instantiates the class as the first step. From there, the application connects to a mediator specified by a URI, and receives a list of base DTDs that the mediator has available. The application opens an available base DTD with one call, and receives a set of three items: 1) a tree data structure representing the source's XML content, 2) a tree data structure representing the source's DTD, and 3) an internal identification value for future transactions. Next, the application constructs an XMAS query using BBQSession calls and the given data structures. Finally, the application calls `executeXMASQuery()`, and is returned a new set of three items, except those items now represent the query result.

As a proof of concept, the BBQ interface discussed in this paper is in fact a client of the BBQ API. Currently, we are laying the groundwork for BBQ API to be used in the engine for a query tool whose domain is the Protein Data Bank. From this proposed project we expect valuable feedback regarding the API's flexibility, completeness and simplicity in different query domains.

5. Current Status and Future Work

Almost all of BBQ's interface has been completed. We are currently working on the Query History modules. There are plans to create an XML browser for a more user-friendly display of XML instances; efforts in that direction would hinge upon a more finalized XSL (or equivalent standard) specification. Also, some policy needs to be devised regarding BBQ updating stale persistent virtual views when their constituent elements change or disappear.

Acknowledgements

---

[3] BBQ API documentation at http://www-cse.ucsd.edu/users/kmunroe/MIX/documentation/docsForMix/index.html

wish to thank Vincent Chu and Andreas Giannakopoulos for their initial work on BBQ 1.0, and Bertram Ludaescher for helpful suggestions on the current version.

References

[AQM+97]   Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, Janet L. Wiener: The Lorel Query Language for Semistructured Data. *Int. J. on Digital Libraries 1(1):* 68-88 (1997) [DBLP:journals/jodl/AbiteboulQMWW97]

[BDS95]   P.Buneman, S.Davidson, and D.Suciu. Programming Constructs for Unstructured Data. In *DBPL-5*, Gubbio, Italy, 1995.

[BDH+96]   Peter Buneman, Susan B. Davidson, Gerd G. Hillebrand, Dan Suciu: A Query Language and Optimization Techniques for Unstructured Data. *SIGMOD Conf. 1996*: 505-516 [DBLP:conf/sigmod/BunemanDHS96]

[BLP+98]   C. Baru, B. Ludaescher, Y. Papakonstantinou, P. Velikhov, and V. Vianu. Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. In http://www.w3.org/TandS/QL/QL98/pp/xmas.html

[CHM+96]   Michael J. Carey, Laura M. Haas, Vivekananda Maganty, John H. Williams. PESTO: An Integrated Query/Browser for Object Databases. *VLDB 1996*: 203-214 [DBLP:conf/vldb/CareyHMW96]

[CKK+99]   S. Cohen, Y. Kanza, Y. Kogan, W. Nutt, Y. Sagiv, and A. Serebrenik. EquiX – Easy Querying in XML Databases. In http://www-rocq.inria.fr/~cluet/WEBDB/procwebdb99.html

[CMH+94]   Sudarshan S. Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, Jennifer Widom: The TSIMMIS Project: Integration of Heterogeneous Information Sources. *IPSJ 1994*: 7-18 [DBLP:conf/ipsj/ChawatheG HPUW94]

[DOM98]   Document Object Model (DOM) 1.0. W3C recommendation, 1998. Available at http://www.w3.org/TR/REC-DOM-Level-1/

[GGK99]   Ignacio Gil, Peter M.D. Gray and Graham J.L. Kemp. A Visual Interface and Navigator for the P/FDM Object Database. *UIDIS99*: 54-63

[GW97]   Roy Goldman, Jennifer Widom: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. *VLDB 1997*: 436-445 [DBLP:conf/vldb/GoldmanW97]

[GW98]   Roy Goldman, Jennifer Widom: Interactive Query and Search in Semistructured Databases. *WebDB 1998*: 52-62 [DBLP:conf/webdb/GoldmanW98]

[IBM99]   VisualDTD. Available at http://www.alphaworks.ibm.com/tech/VisualDTD.

[LPV99]   B. Ludaescher, Y. Papakonstantinou, P. Velikhov. A Framework for Navigation Driven Lazy Mediators, *WebDB 99 Workshop.*

[LPV+99]   B. Ludaescher, Y. Papakonstantinou, P. Velikhov, V. Vianu. View Definition and DTD Inference for XML, *Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats (SSD99)*, Jerusalem, 1999.

[MAG+97]   Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, Jennifer Widom: Lore: A Database

Management System for Semistructured Data. *SIGMOD Record 26(3):* 54-66 (1997) [DBLP:journals/sigmod/McHughAGQW97]

[MAM+98] G. Mecca, P. Atzeni , A. Masci , G. Sindoni , P. Merialdo . The Araneus Web-based management system. *ACM SIGMOD* Record Vol. 27, No. 2 (June 1998), Pages 544-546

[PV99] Y. Papakonstantinou, P. Velikhov. Enhancing Semistructured Data Mediators with Document Type Definitions. *Int. Conf. On Data Engineering,* March 1999.

[Sha95] Keith E. Shafer: Creating DTDs via the GB-Engine and Fred. *SGML 1995*. Accessible at http://www.clc.org/fred/docs/sgml95.html

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38-49, 1992

XML-QL XML-QL: A Query Language for XML. W3C note available at www.w3.org/TR/note-xml-ql

[XML98] Extensible Markup Language (XML) 1.0. W3C recommendation ,1998. Available at http://www.w3.org/TR/REC-xml