A Framework for XML-based Integration of Data, Visualization and Analysis in a Biomedical Domain

N. Bales, J. Brinkley, E. S. Lee, S. Mathur, C. Re, and D. Suciu

University of Washington

Abstract. Biomedical data are becoming increasingly complex and heterogeneous in nature. The data are stored in distributed information systems, using a variety of data models, and are processed by increasingly more complex tools that analyze and visualize them. We present in this paper our framework for integrating biomedical research data and tools into a unique Web front end. Our framework is applied to the University of Washington's Human Brain Project. Specifically, we present solutions to four integration tasks: definition of complex mappings from relational sources to XML, distributed XQuery processing, generation of heterogeneous output formats, and the integration of heterogeneous data visualization and analysis tools.

1 Introduction

Modern biomedical data have an increasingly complex and heterogeneous nature, and are generated by collaborative yet distributed environments. For both technical and sociological reasons these complex data will often be stored, not in centralized repositories, but in distributed information systems implemented under a variety of data models. Similarly, as the data becomes more complex, the tools to analyze and visualize them also become more complex, making it difficult for individual users to install and maintain them.

The problem we are addressing is how to build a uniform Web interface that (a) gives users integrated access to distributed data sources, (b) allows users to formulate complex queries over the data without necessarily being competent in a query language, (c) allows access to existing visualization tools which do not need to be installed on the local workstation, and (d) allows control of existing data analysis tools, both for data generation, and processing of query results.

Our specific application is the integration of data sources containing multimodality and heterogenous data describing language organization in the brain, known as the University of Washington's Human Brain Project [7]. The Web front end is targeted towards sophisticated and demanding users (neuroscience researchers). We examine in this paper the components that are needed to perform such a data integration task, and give a critical assessment of the available XML tools for doing that.

We have identified a few data management problems that need to be addressed in order to achieve integration: complex mappings from relational sources

to XML, distributed XQuery processing, graphical XQuery interfaces, generation of heterogeneous output formats, and integration of data visualization and analysis tools. The main contribution in this paper is to describe our framework for achieving the integration of data, queries, visualization, and analysis tools. Specifically, we make the following contributions:

Complex Relational-to-XML Mapping In our experience, writing complex mappings from relational data to XML data was one of the most labor intensive tasks. We propose a simple extension to XQuery that greatly simplifies the task of writing complex mappings.

Distributed XQuery We identify several weaknesses of the *mediator* model for data integration, and propose an alternative, based on a *distributed query language*. It consists of a simple extension to XQuery, called XQueryD [28], which allows users to distribute computations across sites.

Heterogeneous Output Formats Users want to map the data into a variety of formats, either for direct visualization, or in order to upload to other data processing tools (spreadsheets, statistical analysis tools, etc). We describe a simple interface to achieve that.

Heterogeneous Visualization Tools We propose an approach for integrating multiple data visualization tools, allowing their outputs to be incorporated into query answers.

2 Application Description

The driving application for this work is the University of Washington Integrated Brain Project, the goal of which is to develop methods for managing, sharing, integrating and visualizing complex, heterogeneous and multi-modality data about the human brain, in the hope of gaining a greater understanding of brain function than could be achieved with a single modality alone [7]. This project is part of the national Human Brain Project (HBP) [21], whose long-term goal is to develop interlinked information systems to manage the exploding amount of data that is being accumulated in neuroscience research.

Within the UW HBP the primary data are acquired in order to understand language organization in the brain. Because each type of data is complex, we have developed and are continuously developing independent tools for managing each type, with the belief that each such tool will be useful to other researchers with similar types of data, and with the aim of integrating the separate tools, along with external tools, in a web-based data integration system that relies on XML as the medium of data exchange.

The web-based integration system we are developing is called XBrain [35, 36]. The components that this system seeks to integrate include data sources, visualization tools and analysis tools.

2.1 Data Sources

We describe here three of the data sources in XBrain, which illustrate data stored in three different data models: relational, ontology, and XML.

A relational database: CSM (Cortical Stimulation Mapping) This is a patient-oriented relational database stored in MySQL, which records data obtained at the time of neurosurgery for epilepsy. The data primarily represent the cortical locations of language processing in the brain, detected by noting errors made by the patient during electrical stimulation of those areas. The database also contains the file locations of image volumes, 3-D brain models, and other data needed in order to reconstruct a model of the brain from MRI images, and to use that model as a basis for calculating the locations of the language sites. Data are entered by means of a web-based application [20], but only minimal browse-like queries were supported by the legacy application. The database has 36 tables containing 103 patients, and is 8MB in size.

An ontology: FMA (Foundational Model of Anatomy) This ontology is the product of a separate, major research project conducted over more than ten years at the University of Washington [30]. The FMA is a large semantic network containing over 70,000 concepts representing most of the structures in the body, and 1.2 million relationships, such as `part-of`, `is-a`, etc. The FMA relates to the CSM database through the names of anatomical brain regions where the stimulation sites are located: e.g. FMA could be used to find neighboring, contained, or containing regions of specific stimulation sites in CSM. The FMA ontology is stored and managed in Protege[1], which is a general purpose ontology managing system, and does not support a query language. A separate project [27] built a query interface to FMA, called OQAFMA, which supports queries written in StruQL [15], a query language specifically designed for graphs.

An XML File: IM (Image Manager) As part of an anatomy teaching project we have developed a tool for organizing teaching images [5]. Each image has associated with it one or more annotation sets, consisting of one or more annotations. An annotation consists of a closed polygon specified by a sequence of image coordinates on the image and an anatomical name describing that region. As in the CSM database, the names are taken from the FMA. In the original project the data is stored in a relational database. For the purpose of integrating it in XBrain we converted it into a single XML document, because it is infrequently updated because it has a natural recursive structure. To query it, we use the Galax [13] XQuery interpretor.

2.2 Visualization Tools

Many tools for Web-based visualization and interaction with both 2-D and 3-D images have been developed, in our lab and elsewhere. For example, we have developed interactive tools for 2-D images [6], and tools for 3-D visualization of CSM and other functional language data mapped onto a 3-D model of a patient or population brain [26]. These tools are being integrated as part of the XBrain project. While each tool is designed to display a single image at a time, in XBrain we allow users to integrate images generated by several visualization tools with the data returned by queries.

[1] `http://protege.stanford.edu/`

2.3 Analysis Tools

Finally, users are sophisticated, and they generally develop or use various analysis tools to generate the data that are entered into the various data sources, or to further process the results of a query. An example tool for the UW HBP is the Visualization Brain Mapper (VBM) [19], which accepts a specification file generated from the CSM database, then creates a mapping of stimulation sites onto a generated 3-D model. A second example is our X-Batch program [18] which provides a plugin to a popular functional image analysis program while transparently writing to a backend database. These tools are being integrated into XBrain, by having queries generate appropriate data formats for them.

2.4 The Problem

The UW HBP data sources and tools illustrate the increasingly complex and heterogenous nature of modern biomedical data, as well as the increasingly collaborative yet distributed environment in which they are generated. These complex data are stored, not in a centralized repository, but in distributed information systems implemented under a variety of data models. The tools to analyze and visualize them are also quite complex, making it difficult for individual users to install and maintain them. Thus, the problem we are addressing is how to build a uniform Web interface that (a) gives users integrated access to distributed data sources, (b) allows users to formulate complex queries over the data without necessarily being competent in a query language, (c) allows access to existing visualization tools which do not necessarily need to be installed on the local workstation, and (d) allows control of existing data analysis tools, both for data generation, and processing of query results. XBrain is our proposed framework for addressing these problems.

3 The XBrain Integration Architecture

Our architecture is shown in Fig. 1. All sources store data in their native format and have to map the data to XML when exported to the query processor. Most mapped sources accept XQuery over their data, with one exception: OQAFMA accepts StruQL queries, because the rich structure of the ontology describing all of human anatomy requires a richer language than XQuery for recursive path traversals in the ontology graph. The data from all sources are integrated by a module supporting a distributed extension of the XQuery language (XQueryD). This module sends queries to the local sources and integrates the resulting XML data fragments. The resulting XML query answer can be presented to the user in one of multiple formats: as a plain XML file, as a CSV (Comma Separated Values) file, or a nested HTML file. In the latter case, the image anchors embedded in the XML file are interpreted by calling the appropriate image generation Webservices, and the resulting HTML pages together with complex images is presented to the user. The user inputs queries expressed in XQueryD through a JSP page. We describe the specific data management tasks next.

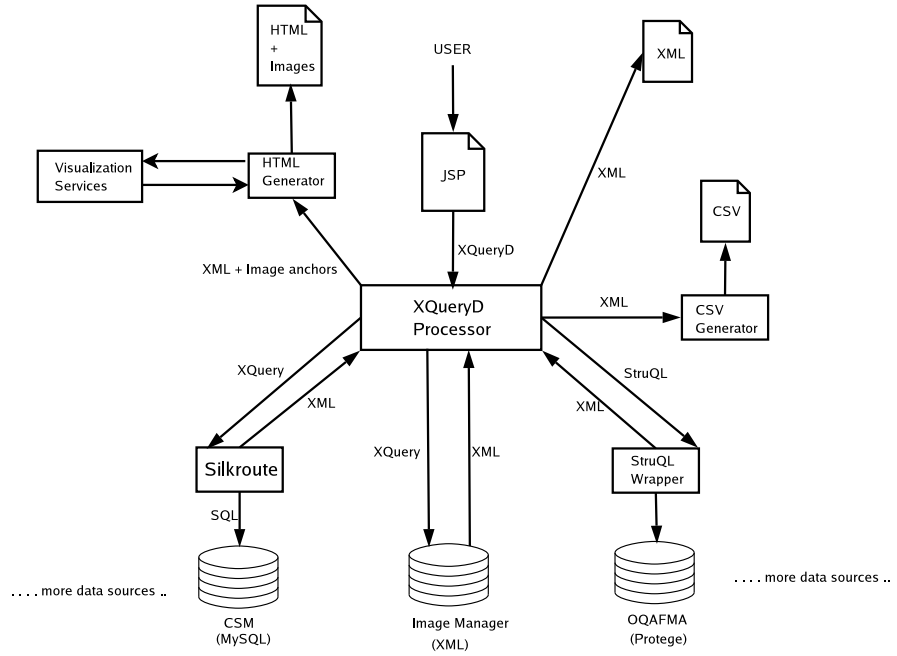


Fig. 1. The XBrain Integration Architecture.

4 Specific Data Management Tasks

4.1 Mappings to XML

We mapped the relational CSM database to XML using SilkRoute [11, 35]. To define the map, one needs to write an XQuery program that maps the entire relational database to a virtual XML document, called the *public view*. Users query this view using XQuery, which SilkRoute translates to SQL, then converts the answers back to XML. For example, the user query below finds the names of all structures over all patients, in which a CSM error of type 2 (semantic paraphasia) occurred at least once in one patient:

```
<results>
 {for $trial in PublicView("Scrubbed.pv")/patient/surgery/csmstudy/trial
  where $trial/trialcode/term/abbrev/text()="2"
  return $trial/stimsite/name()
 }
</results>
```

Here `PublicView` indicates the file containing the public view definition (a large XQuery). SilkRoute translates the query automatically into a complex SQL statement:

```
  SELECT stimsite.name
  FROM  trial, csm, . . ., stimsite
  WHERE term.type = 'CSM error code' AND abbrev = '2' AND  . . .
```

```
ExprSingle    ::= TblExpr | ... (* all expressions in XQuery remain here *)

TblExpr       ::= NameClause WhereClause? OmitClause? RenameClause? ReturnClause?

NameClause    ::= "table " TblName (" as " <NCName>)?
OmitClause    ::= "omit " ColName (", " ColName)*
RenameClause  ::= "rename " ColName as <NCName> (", " ColName as <NCName>)*
ReturnClause  ::= "return " EnclosedExpr

TblName       ::= <NCName>
ColName       ::= <NCName>
FunctionCall  ::= <QName "("> (ExprSingle ("," ExprSingle)*)? ")" ("limit" IntegerLiteral)?
```

Fig. 2. The Grammar for RXQuery

Writing the public view was a major task. For XBrain, it had 583 lines of XQuery code, which was repetitive, boring to write, and error prone. We needed a more efficient tool to write such mappings, in order to easily extend XBrain to other sources. For that, we developed a simple extension to XQuery that allows the easy specification of complex mappings from relational data to XML.

RXQuery: A Language for Mapping Relations to XML Our new language allows users to concisely specify complex mappings from relational databases to XML such that (1) default mappings are done automatically, by using the relational database schema, and (2) the user can override the defaults and has the full power of XQuery. The grammar is shown in Fig. 2. It extends the XQuery syntax with seven new productions, by adding "table expressions", `TblExpr` to the types of expressions in the language. The RXQuery preprocessor takes as input a relational database schema and an RXQuery expression and generates an XQuery expression that represents a public view.

Example 1. We illustrate RXQuery with three examples, shown in Fig. 3. In all of them we use a simple relational database schema consisting for the two relations below, which are a tiny, highly simplified fragment of CSM:

```
patient(pid, name, dob, address)
surgery(sid, pid, date, surgeon)
```

Consider $Q1$ in Fig. 3 and its translation to XQuery. By default, every column in the relational schema is mapped into an XML element with the same name. Here `CanonicalView()` is a SilkRoute function that represents the canonical view of the relational database.

Query $Q2$ illustrates the `omit` and the `rename` keywords that omit and/or rename some of these attributes, and the `where` and the `return` clauses that allow the user to restrict which rows in the table are to be exported in XML and to add more subelements to each row. In $Q2$, the `name` column is omitted, the `dob` column is exported as the `@date-of-birth` attribute, rather than the default `dob` element, and an additional element `age` is computed for each row.

RXQuery is especially powerful when specifying complex, nested public views, which is the typical case in practice. $Q3$ is a very simple illustration of this power.

	RXQuery	Translation to XQuery

$Q1$	`table patient`	`for $Patient in CanonicalView()/patient`
		`return`
		`  <patient>`
		`  <pid> { $Patient/pid/text() } </pid>`
		`  <name> { $Patient/name/text() } </name>`
		`  <dob> { $Patient/dob/text() } </dob>`
		`  <address> { $Patient/address/text() } </address>`
		`  </patient>`

$Q2$	`table patient`
	`  omit name`
	`  rename dob as @date-of-birth`
	`where $Patient/dob/text() < 1950`
	`return`
	`  <age>`
	`  { 2005 - $Patient/dob/text() }`
	`  </age>`
		`for $Patient in CanonicalView()/patient`
		`where $Patient/dob/text() < 1950`
		`return`
		`  <patient date-of-birth = ‘‘$Patient/dob/text()’’>`
		`  <pid> { $Patient/pid/text() } </pid>`
		`  <address> { $Patient/address/text() } </address>`
		`  <age> { 2005 - $Patient/dob/text() } </age>`
		`  </patient>`

$Q3$	`table patient`
	`return`
	`  table surgery omit pid`
	`  where $Patient/pid/text() =`
	`  $Surgery/pid/text()`
		`for $Patient in CanonicalView()/patient`
		`return`
		`  <patient>`
		`  <pid> { $Patient/pid/text() } </pid>`
		`  <name> { $Patient/name/text() } </name>`
		`  <dob> { $Patient/dob/text() } </dob>`
		`  <address> { $Patient/address/text() } </address>`
		`  { for $Surgery in CanonicalView()/surgery`
		`  where $Patient/pid/text() = $Surgery/pid/text()`
		`  return <surgery>`
		`  <sid> $Surgery/sid/text() </sid>`
		`  <date> $Surgery/date/text() </date>`
		`  <surgeon> $Surgery/surgeon/text()`
		`  </surgeon>`
		`  </surgery>`
		`  </patient>`

Fig. 3. Examples of RXQuery and their translations to XQuery.

Here, the nested subquery is a simple `table` expression, which is expanded automatically by the preprocessor into a complex subquery.

In addition to the features illustrated in the example, RXQuery includes functions, which we have found to be important in specifying complex mappings, since parts of the relational database need to be included several times in the XML document. The `limit n` clause (see Fig. 2) represents a limit on the recursion depth, when the function is recursive: this allows us some limited form recursive XML views over relational data (SilkRoute does not support recursive XML structures).

One measure of effectiveness of RXQuery is its conciseness, since this is correlated to the readability and maintainability of the public views. Fig. 4 reports the number of lines for two public views: for CSM and for the original version of IM (which is in a relational database). The CSM public view became about 5 times smaller, shrinking from 583 lines in XQuery to 125 lines in RXQuery. The IM public view shrank from an original XQuery with 1383 lines of code to an RXQuery expression with only 151 lines. In both examples, the XQuery public view generated automatically by the RXQuery preprocessor was only sightly larger than the original manual public view.

PV for CSM:

Public View	Lines	Words	Chars
XQuery(manual)	583	1605	28582
RXQuery(w/o functions)	141	352	4753
RXQuery(w/ functions)	125	303	4159
XQuery(generated)	634	1633	34979

PV for IM:

Public View	Lines	Words	Chars
XQuery(manual)	1383	3419	64393
RXQuery(w/o functions)	381	1178	14987
RXQuery(w/ functions)	151	427	5603
XQuery(generated)	1427	3575	66105

Fig. 4. Two examples of large public views defined in RXQuery: on the CSM database and on the original relational version of the IM (Image Manager) database. The tables show the original, manual definition of the public view in XQuery, the definition in RXQuery without functions, the same with functions, and the resulting, automaticallly generated XQuery.

Mapping Other Data Sources to XML While most data sources can be mapped to XML in a meaningful way, sometimes this is not possible. In such cases we decided to keep the original data model, rather than massaging it to an artificial XML structure. The Foundational Model of Anatomy (FMA) is a rich ontology, which is best represented as a graph, not a tree. We kept its query interface, OQAFMA, which uses StruQL as a query language and allows users to express complex recursive navigation over the ontology graph. For example, the StruQL query below returns all anatomical parts that contain the middle part of the superior temporal gyrus:

```
WHERE  Y->":NAME"->"Middle part of superior temporal gyrus",
       X->"part"*->Y,
       X->":NAME"->Parent
CREATE  Concept(Parent);
```

The query computes a transitive closure of the `part` relationship. While the query data model is best kept as a graph, the query answers can easily be mapped back into XML. In our example, the answer returned by OQAFMA is:

```
<results> <Concept> <Ancestor>Neocortex</Ancestor>     </Concept>
          <Concept> <Ancestor>Telencephalon</Ancestor> </Concept>
  . . . . . .
</results>
```

Finally, native XML data are queried directly using XQuery. In our case, the Image Manager data (IM) is stored in XML and queried using Galax [13]. The following example finds all images annotated by the *middle part of the superior temporal gyrus*:

```
for $image in document("image_db.xml")//image
where $image/annotation_set/image_annotation/name/text() =
                                  "middle part of the superior temporal gyrus"
return <image>  {$image/oid}  </image>
```

4.2 Distributed XQuery Processing

The standard approach to data integration is based on a mediator, an architecture proposed by Gio Wiederhold [38]. With this approach, a single mediator schema is first described over all sources, and all local sources are mapped into the mediated schema.

```
ExprSingle   ::= "execute at" <URL>  [ "xquery" { ExprSingle }  |  "foreign" { String } ]
                    ( "handle" <VAR>:<NAME-SPACE> <EXPR> )*
```

Fig. 5. Grammar for XQueryD

We found this approach too heavy duty for our purpose, for three reasons. First, mediators are best suited in cases when the same concept appears in several sources, and the mediated concept is the set union of the instance of that concept at the sources. For example, BioMediator, a mediator-based data integration project for genetic data [32], integrates several sources that have many overlapping concepts: e.g. most sources have a `gene` class, and the mediator defines a global `gene` class which is the logical union of those at the local sources; similarly, most sources have a `protein` concept, which the mediator also unions. By contrast, in XBrain the concepts at the sources are largely disjoint, and the mediated schema would trivially consist of all local schemas taken together, making the mediator almost superfluous.

The second reason is that mediator based systems require a unique data model for all sources, in order to be able to perform fully automatic query translation. They also hide the schema details at sources from the user, allowing inexperienced users to access large numbers of data sources. None of these applies to XBrain: some sources (like FMA) are best kept in their native datamodel, which is not XML, and our sophisticated users are quite comfortable with the details of the source schemas.

Finally, despite fifteen years of research, there are currently no widely available, robust tools for building mediator systems.

Our approach in XBrain is different, and is based on a distributed evaluation of XQuery. All local sources are fully exposed to the users, who formulate XQuery expressions over them.

XQueryD: A Distributed XQuery Language The goal is to allow users to query multiple sources in one query. While this can already be done in XQuery, it supports only the *data shipping* model (through the `document()` function): it fetches all data sources to a single server, then runs the query there. This is a major limitation for many applications, especially when some data sources are very large, or when a data source is only a virtual XML view over some other logical data model. For example, our CSM data source is not a real XML document, but a virtual view over a relational database. If we materialized it, the 8MB relational database becomes a 30MB XML document; clearly, it is very inefficient to fetch the entire data with `document()`. We propose a simple extension to XQuery that allows *query shipping* to be expressed in the language, in addition to data shipping. The language consists of a single new construct added to `ExprSingle`, and is shown in Fig. 5

Example 2. We illustrate XQueryD with one single example. The query in Fig. 6 integrates three sources: the Image database (XML), the CSM databases (rela-

```
for $image in document("image_db.xml")//image
let $region_name := execute at "http://csm.biostr.washington.edu/axis/csm.jws"
      xquery { for $trial in PublicView("Scrubbed.pv")/patient/surgery/csmstudy/trial
               where $trial/trialcode/term/abbrev/text()="2"
               return $trial/stimsite/name()
             },
    $surrounding_regions :=
        for $term in $region_name
        return <term> {(execute at "http://csm.biostr.washington.edu/oqafma"
                        foreign  {WHERE Y->":NAME"->"$term",
                                        X->("part")*->Y,
                                        X->":NAME"->Ancestor
                                  CREATE  Concept(Ancestor); }
                        )/results/Concept/text()
                      }
                </term>
where $image/annotation_set/image_annotation/name/text() = $surrounding_regions/text()
return $image/oid/text()
```

Fig. 6. Example of a query in XQueryD

tional), and the OQAFMA database (ontology). The query starts at the image database, and iterates over all images. The `execute at` command instructs the query processor to send the subsequent query to a remote site for execution: there are two remote queries in this example. The query returns all images are annotated by an anatomical name that is part of the anatomical region surrounding any language site with error type 2.

We initially implemented XQueryD by modifying Galax to accept the new constructs, which was a significant development effort. Once new versions of Galax were released, we found it difficult to keep up with Galax' code evolution. We are currently considering implementing a translator from XQueryD to XQuery with Webservice calls that implement the `execute` statements. For the future, we argue for the need of a standard language extension of XQuery to support distributed query processing in query shipping mode.

Discussion There is a tradeoff between the mediator-based approach and the distributed query approach. In XQueryD users need to know the sources' schemas, but can formulate arbitrarily complex queries as long as these are supported by the local source. Adding a new source has almost no cost. A mediator based systems presents the user with a logically coherent mediated schema, sparing him the specific details at each source; it can potentially scale to large number of sources. On the other hand, users can only ask limited form of queries supported by the mediator, typically conjunctive queries (i.e. without aggregates or subqueries), and the cost of adding a new source is high.

4.3 Graphical Query Interface

In order to allow easy access to the integrated sources and to all data processing tools, XBrain needs to allow users to formulate complex queries over the data

without necessarily being competent in a query language. Our current approach is to provide the user with (a) a free form for typing XQueryD expressions, (b) a number of predefined XQueryD expressions, which can be modified by the users in free form, and (c) a simple interface that allows users to save query expressions and later retrieve and modify them. This part of the system will be extended in the future with elements of graphical query languages; there is a rich literature on graphical query interfaces, e.g. QBE [40] and XQBE [4].

4.4 Heterogeneous Output Formats

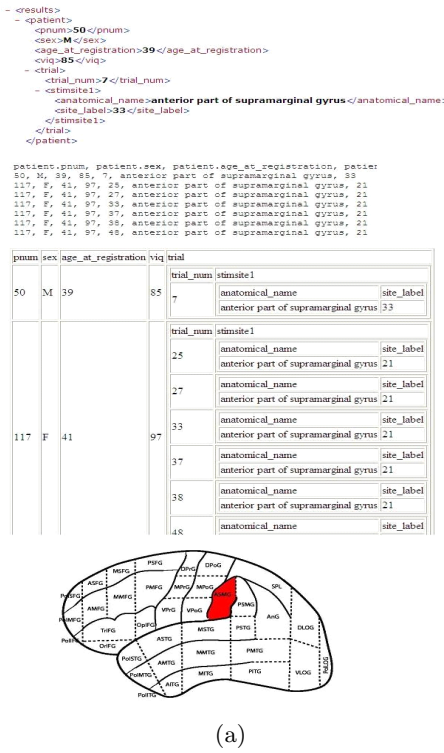
The output of XQueryD is a single XML document describing the results of integrating data from the different data sources. While such a document may be useful for analysis programs or XML-savvy users, it is not the most intuitive. Thus, we allow users to choose alternative output formats, including both common formats such as HTML or CSV (comma separated values for input to Excel), and formats that are specific for an application.

In our approach, the system generates automatically an XSLT program for each XQueryD, and for each desired output format. The XSLT program is simply run on the query's answer and generates the desired output format. We currently support CSV, HTML, and some proprietary formats for image generation tools. To generate the XSLT program, the system needs to know the structure of the XML output, i.e. the element hierarchy and the number of occurrences of each child, which can be `*`, `1`, or `?` (0 or 1). In an early version we computed this structure by static analysis on the query (type inference), but we found that code brittle and hard to maintain and are currently extracting the structure from the XML output: the tiny performance penalty is worth the added robustness. Figure 7 (a) shows four possible output formats for the same output data: in XML format, in CSV format, in HTML format, and as an image.

4.5 Integration of Heterogeneous Visualization Tools

Common output transformations, such as HTML or CSV as noted in the previous section, can be part of a generic integrated application that could be applied to many different problems. However, each biomedical application will have its own special output requirements that may best be addrsseed by independent visualization tools. Our approach to this problem is to create independent tools that can run as web services callable by the XBrain application. We have experimented with such services for a 2-D image visualization tool that accepts XML output from the CSM database and generates an image showing the locations on a 2-D sketch of the brain where specific types of language processing occur.

Such an approach may also be useful for 3-D visualization of query results, using a server-based version of our BrainJ3D visualization tool [26]. Interactive visualization and analysis of the results, which might include new query formation, will require alternative approaches, such as a Web Start application that can create an XQuery for the integrated query system.



```
- <results>
  - <patient>
      <pnum>50</pnum>
      <sex>M</sex>
      <age_at_registration>39</age_at_registration>
      <viq>85</viq>
    - <trial>
        <trial_num>7</trial_num>
      - <stimsite1>
          <anatomical_name>anterior part of supramarginal gyrus</anatomical_name>
          <site_label>33</site_label>
        </stimsite1>
      </trial>
    </patient>
```

```
patient.pnum, patient.sex, patient.age_at_registration, patier
50, M, 39, 85, 7, anterior part of supramarginal gyrus, 33
117, F, 41, 97, 25, anterior part of supramarginal gyrus, 21
117, F, 41, 97, 27, anterior part of supramarginal gyrus, 21
117, F, 41, 97, 33, anterior part of supramarginal gyrus, 21
117, F, 41, 97, 37, anterior part of supramarginal gyrus, 21
117, F, 41, 97, 38, anterior part of supramarginal gyrus, 21
117, F, 41, 97, 48, anterior part of supramarginal gyrus, 21
```

pnum	sex	age_at_registration	viq	trial
50	M	39	85	trial_num: 7; stimsite1: anatomical_name = anterior part of supramarginal gyrus, site_label = 33
117	F	41	97	trial_num: 25; stimsite1: anatomical_name = anterior part of supramarginal gyrus, site_label = 21
				trial_num: 27; stimsite1: anatomical_name = anterior part of supramarginal gyrus, site_label = 21
				trial_num: 33; stimsite1: anatomical_name = anterior part of supramarginal gyrus, site_label = 21
				trial_num: 37; stimsite1: anatomical_name = anterior part of supramarginal gyrus, site_label = 21
				trial_num: 38; stimsite1: anatomical_name = anterior part of supramarginal gyrus, site_label = 21
				trial_num: 48; stimsite1: anatomical_name = anterior part of supramarginal gyrus, site_label

(a)

```
<results>
  { for $trial in PublicView("Scrubbed.pv")
      /patient/surgery/csmstudy/trial
    where $trial/trialcode/term/abbrev/text()="2"
    return
      <answer>
                <name>
                  { $trial/stimsite/name() }
                </name>
                <image-anchor>
                  <uri>
                    http://csm.biostr.washington.edu/vbm
                  </uri>
                  <param>
                    { $trial/stimsite/name() }
                  </param>
                  <param>
                    gray
                  </param>
                </image-anchor>
      </answer> }
</results>
```

(b)

Fig. 7. Different query output formats (XML, CSV, HTML, and Image) (a). Query for embedding images in XML files (b).

Our approach is as follows (we refer to Fig. 1 below). Every visualization tool must be a Webservice and offer a common interface that accepts some tool specific input parameters and generates an image (jpeg file). The XQueryD expression returns certain subelements in the answer that are *anchors* to visualization Webservices. The user has to explicitly construct these anchors, like in Fig. 7 (b), which returns a set of stimulation sites, each with an image of the brain with the corresponding stimulation site highlighted. The answer to the query contain elements `image-anchor`. When the HTML generator (see Fig. 1) translates the XML document into HTML, it processes the image anchors by calling the appropriate Webservice with the appropriate parameters, then embeds the resulting images in the HTML table.

5 Related Work

Distributed Query Processing There is a rich literature on distributed query processing and optimization; a survey is in [22]. Our syntactic approach to distributed query is closest in spirit to the Kleisli system [39], and also re-

lated to process calculi and their application to database queries [31, 17]. Unlike ubQL [31], we use only one communication primitive, namely the migration operator `execute`, and omit channels and pipelining. A different approach to distributed data is Active XML [3, 2, 1]. Here an XML tree may contain calls to Webservices that return other XML fragments. Query evaluation on Active XML naturally leads to distributed execution. XQueryD differs from Active XML in several ways. In Active XML the data need to be modified by inserting Web-service calls and users are unaware of the distributed nature of the data; by contrast, in XQueryD the data do not need to be modified, while queries require detailed knowledge of the sources and their capabilities.

Mapping Relational Data to XML Mapping relational data to XML has been discussed extensively [14, 8, 34, 33, 37, 12]. In addition, most of the database vendors today offer some support for XML publishing: Oracle [24, 10], SQL Server [25], BEA's Liquid Data [9]. Each mapping language is proprietary, and can only be used in conjunction with that particular product (relational database or query engine). In addition, none offers any shortcuts to defining the mapping: users have to write each piece of the mapping. In contrast RXQuery is more lightweight, and is translated into XQuery, and is specifically designed to be very concise when defining complex mappings.

Other Related Work For XQuery processing we use the Galax, which is described in [13]. For translating XQuery to SQL we use SilkRoute, whose architecture is described in [11], and which we recently extended significantly to generated optimized SQL code: the extensions and optimizations are described in [29]. Other optimization techniques for the XQuery to SQL translation are discussed in [23]. There is a rich literature on graphical query interfaces, starting with Zloof's QBE [40]. Recent work describes a graphical query interface to XQuery, called XQBE, is in [4].

6 Conclusions

The high complexity in integrating today's biomedical data has two root causes: the fact that the data are increasingly distributed and generated by collaborative environments, and the fact that they are processed, analyzed and visualized by increasingly more complex tools. We have described a framework for integrating data and tools for biomedical data, with a specific application to the University of Washington's Human Brain Project.

Acknowledgments: This work was funded in part by NIH Human Brain Project grant DC02310 and Suciu was partially supported by the NSF CAREER Grant IIS-0092955, NSF Grants IIS-0140493, IIS-0205635, and IIS-0428168, and a gift from Microsoft.

References

1. S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy query evaluation for active XML. In *SIGMOD*, 2004.
2. S. Abiteboul, O. Benjelloun, and T. Milo. Positive active xml. In *PODS*, 2004.
3. S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic xml documents with distribution and replication. In *SIGMOD*, pages 527–538, 2003.
4. E. Augurusa, D. Braga, A. Campi, and S. Ceri. Design of a graphical interface to XQuery. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 226–231, 2003.
5. J. Brinkley, R. Jakobovits, and C. Rosse. An online image management system for anatomy teaching. In *Proc. AMIA Fall Symposium*, page 983, 2002.
6. J. Brinkley, B. Wong, K. Hinshaw, and C. Rosse. Design of an anatomy information system. *Computer Graphics and Applications*, 19(3):38–48, 1999. Invited paper.
7. J. F. Brinkley, L. M. Myers, J. S. Prothero, G. H. Heil, J. S. Tsuruda, K. R. Maravilla, G. A. Ojemann, and C. Rosse. A structural information framework for brain mapping. In *Neuroinformatics: An Overview of the Human Brain Project*, pages 309–334. Mahwah, New Jersey: Lawrence Erlbaum, 1997. See also `http://sig.biostr.washington.edu/projects/brain/`.
8. M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. subramanian. XPERANTO: publishing object-relational data as XML. In *Proceedings of WebDB*, Dallas, TX, May 2000.
9. M. J. Carey. BEA liquid data for WebLogic: XML-based enterprise information integration. In *ICDE*, pages 800–803, 2004.
10. A. Eisenberg and J. Melton. SQL/XML is making good progress. *SIGMOD Record*, 31(2):101–108, 2002.
11. M. Fernandez, Y. Kadiyska, A. Morishima, D. Suciu, and W. Tan. SilkRoute : a framework for publishing relational data in XML. *ACM Transactions on Database Technology*, 27(4), December 2002.
12. M. Fernandez, A. Morishima, and D. Suciu. Efficient evaluation of XML middleware queries. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Santa Barbara, 2001.
13. M. Fernandez and J. Simeon. Galax: the XQuery implementation for discriminating hackers, 2002. available from `http://db.bell-labs.com/galax/`.
14. M. Fernandez, D. Suciu, and W. Tan. SilkRoute: trading between relations and XML. In *Proceedings of the WWW9*, pages 723–746, Amsterdam, 2000.
15. M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. Declarative specification of web sites with strudel. *VLDB Journal*, 9(1):38–55, 2000.
16. J. Funderburk, G. Kiernan, J. Shanmugasundaram, E. Shekita, and C. Wei. Technical note - XTABLES: Bridging relational technology and XML. *IBM Systems Journal*, 42(3):538–, 2003.
17. P. Gardner and S. Maffeis. Modelling dynamic Web data. In *Proceedings of DBPL*, pages 75–84, Potsdam, Germany, 2003.
18. X. Hertzenberg, A. Poliakov, D. Corina, G. Ojemann, and J. Brinkley. X-batch: Embedded data management for fmri analysis. In *Society for Neuroscience Annual Meeting*, page 694.21, San Diego, 2004.
19. K. Hinshaw, A. Poliakov, R. Martin, E. Moore, L. Shapiro, and J. Brinkley. Shape-based cortical surface segmentation for visualization brain mapping. *Neuroimage*, 16(2):295–316, 2002.

20. R. Jakobovits, C. Rosse, and J. Brinkley. An open source toolkit for building biomedical web applications. *J Am Med Ass.*, 9(6):557–590, 2002.
21. S. Koslow and S. Hyman. Human brain project: A program for the new millenium. *Einstein Quarterly J. Biol. Med.*, 17:7–15, 2000.
22. D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.
23. R. Krishnamurthy, R. Kaushik, and J. Naughton. Efficient XML-to-SQL query translation: Where to add the intelligence? In *VLDB*, pages 144–155, 2004.
24. M. Krishnaprasad, Z. Liu, A. Manikutty, J. Warner, V. Arora, and S. Kotsovolos. Query rewrite for XML in oracle XML DB. In *VLDB*, pages 1122–1133, 2004.
25. M. Library. Creating xml views by using annotated xsd schemas, 2005.
26. E. Moore, A. Poliakov, and J. Brinkley. Brain visualization in java3d. In *Proceedings, MEDINFO*, page 1761, San Francisco, CA, 2004.
27. P. Mork, J. F. Brinkley, and C. Rosse. OQAFMA querying agent for the foundational model of anatomy: a prototype for providing flexible and efficient access to large semantic networks. *J. Biomedical Informatics*, 36(6):501–517, 2003.
28. C. Re, J. Brinkley, K. Hinshaw, and D. Suciu. Distributed XQuery. In *Workshop on Information Integration on the Web (IIWeb)*, pages 116–121, September 2004.
29. C. Re, J. Brinkley, and D. Suciu. Efficient publishing of relational data to XML. submitted.
30. C. Rosse and J. L. V. Mejino. A reference ontology for bioinformatics: the foundational model of anatomy. *Journal of Bioinformatics*, 36(6):478–500, 2003.
31. A. Sahuguet and V. Tannen. ubQL, a language for programming distributed query systems. In *WebDB*, pages 37–42, 2001.
32. R. Shaker, P. Mork, J. Brockenbrough, L. Donelson, and P. Tarczy-Hornoch. The biomediator system as a tool for integrating biologic databases on the web. In *Proc. Workshop on Information Integration on the Web, held in conjunction with VLDB*, 2004.
33. J. Shanmugasundaram, , J. Kiernana, E. Shekita, C. Fan, and J. Funderburk. Querying XML views of relational data. In *Proceedings of VLDB*, pages 261–270, Rome, Italy, September 2001.
34. J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. In *Proceedings of VLDB*, pages 65–76, Cairo, Egypt, September 2000.
35. Z. Tang, Y. Kadiyska, H. Li, D. Suciu, and J. F. Brinkley. Dynamic XML-based exchange of relational data: application to the Human Brain Project. In *Proceedings, Annual Fall Symposium of the American Medical Informatics Association*, pages 649–653, Washington, D.C., 2003. http://quad.biostr.washington.edu:8080/xbrain/index.jsp.
36. Z. Tang, Y. Kadiyska, D. Suciu, and J. Brinkley. Results visualization in the xbrain xml interface to a relational database. In *Proceedings, MEDINFO*, page 1878, San Francisco, CA, 2004.
37. I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *SIGMOD*, May 2002.
38. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.
39. L. Wong. The functional guts of the Kleisli query system. In *Proceedings of ICFP*, pages 1–10, 2000.
40. M. M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.