# An annotation database for multimodal scientific data

Cristina Bogdanschi Simone Santini

Escuela Politécnica Superior, Universidad Autónoma de Madrid
Madrid, Spain

## ABSTRACT

In many collaborative research environments novel tools and techniques allow researchers to generate data from experiments and observations at a staggering rate. Researchers in these areas are now facing the strong need for querying, sharing and exchanging these data in a uniform and transparent fashion. However, due to the nature of the various types of heterogeneous data and lack of local and global database structures, standard data integration approaches fail or are not applicable. A viable solution to this problem is the extensive use of metadata. In this paper we present the model of an annotation management system suitable for such research environments, and discuss some aspects of its implementation. Annotations provide rich linkage structure between data and between themselves that translates in a complex graph structure of which annotations and data are the nodes. We show how annotations are managed and used for data retrieval and outline some of the query techniques used in the system.

## 1. INTRODUCTION

In many areas of modern science, scientists need to manage a large and rapidly expanding reservoir of information. The management of all this information is becoming a particularly complex problem due to the intrinsic structural complexity of these data, the large number of remote data sources on which the data are stored, and the variety of data formats in which they come.

In addition to the inherent complexity of scientific data management, the significant amount of scientific information that takes the form of *annotations* rises new data management issues.[1] Annotations are often the first line of development of science, the place where science truly *happens*, in a rough, dynamic, ever changing form. Long before ideas become hypotheses and, after that, published theories, they exist in annotations. For all scientist, but mainly for those engaged in collaborative studies, accessing this wide, informal, and constantly changing reservoir of information can be crucial. Traditionally, annotations were written by a scientist in her laboratory notebook; more recently, annotations have been stored on a file in the scientist's computer. Given the importance of the information contained in the annotations and their increasing number, it would be useful to place them in a database, so that they can be searched and related to each other and to the data to which they refer. The nature of annotations makes this task a complex one.[2] Annotations are semi-structured data that, in addition to referring to the primary data, can refer to each other resulting in a complex *annotation graph*. This graph structure compounds the internal structure of the annotations and the heterogeneity of the data in making annotation management quite a formidable problem.

The interest in data annotation is not entirely new but, in general, researchers have investigated the problem under the assumption that the annotated data proceed from databases with uniform data types,[3] and that the annotations are atomic: when an annotation is associated to a datum, it is understood that all of its content is associated to that datum.

The purpose of this work is a preliminary study and implementation of an annotation system that attempts to overcome these limitations. We propose a model where an annotation is not atomic and consequently, where parts of an annotation can be associated to a datum. We will also take into consideration the variety of data sources and formats that exist today, and include techniques suitable for referring to data from different data sources. Finally, we will define a query language for the graph structure generated by the relations between annotations and those between annotations and data.

Query languages for graphs have been studied to a fair extent, and quite a few of them have been proposed in the research literature.[4–7] In this work, we use an extension to acyclic graphs of regular expressions, whose characteristics have been studied in connections to trees,[8] and that can be evaluated efficiently on acyclic graphs.[9]

Multimedia Content Access: Algorithms and Systems III, edited by Raimondo Schettini,
Ramesh C. Jain, Simone Santini, Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 7255, 72550G

## 2. THE ANNOTATION MODEL

In their work,[10] Srivastava and Velegrakis proposed the use of a relational model for both primary data and annotations. The general idea is that an annotation is a record that is connected to the data it refers to through a relational query, and the query is stored together with the annotation in a relational table. Suppose we have an annotation regarding patients diagnosed with diabetes whose information is stored in the local relational data base. The link between this annotation and its data is the query

```
select patient
from   patient_table
where  diagnosis=diabetes
```

The model has three main limitations:

**i)** All data are relational and come from the same source. The queries that constitute the links do not contain any indication of the source of the data (it is implicit) nor of the query language to be used to send the query and the formalism to receive the results (this also is implicit).

**i)** The model doesn't include relations between annotations: an annotation cannot refer to other annotations.

**iii)** Possibly the most important limitation of the model is that the annotations are *atomic*, since every annotation is a record in a database. That is, while Srivastava and Velegrakis allow associating an annotation to part of a record (through the use of projection operators in the query), it is impossible to associate part of an annotation to some data: the annotation cannot be broken into parts that are individually associated to different data elements.

The annotation data model that we present in this paper is borne out of the desire to overcome these limitations.

A first improvement is the integration of data from a variety of sources. Not all the databases are relational and not all the information is stored in a database. We want our mechanism to be able to query different data sources like the Web, relational and non-relational databases, ontologies, etc.

A second improvement–and one of the most important that can be made to the model proposed in[3]–is to consider the annotations as complex objects, so that a link can originate from a part of an annotation. In the Srivastava annotation model, as we have mentioned, an annotation can link to whole records as well as to record fragments, but a link can't originate from a part of an annotation. Connecting annotation parts to data result in a robust annotation structure, allows more precise annotation definitions. We will consider an annotation as composed of *sections*, which can also contain sections and so on in a recursive fashion. That is, the abstract model of the annotation is a tree. Each node of the tree can contain a link to data, the semantics being that the whole sub-tree that originates at that node is linked to the data.

A third issue that we will consider is the relation between annotations, which contributes crucially to the correct interpretation of the data. Consider, as an example, an annotation that after a period of time doesn't express with accuracy the data it refers to. Sometimes it is not desirable to remove the annotation from the system (since it could still be of interest), but one needs to have a more recent one alongside it, one that clarifies "this annotation refers to annotation X and to the data therein linked".. This can be done by creating a link between these annotations, specifying if the new annotation eith adds information to or supersedes the old one.

An annotation object may connect one or more fragments of one or more data objects. An annotation is therefore a *linker object* that connects the annotation's contents to the annotation's referent(s). In our case, the annotation referent can be not only the object data fragments, but another annotation or annotation part as well. Moreover, the linker object could connect all the annotation content or just part of it to the annotation referent (fig. 1). The model assumes that links can go from annotations to annotations and from annotations to data, but not from data to annotations. That is, the primary data can be pointed to, but can't point. Some of these data elements (such as web pages) can contain links, but these are not annotation links of the type

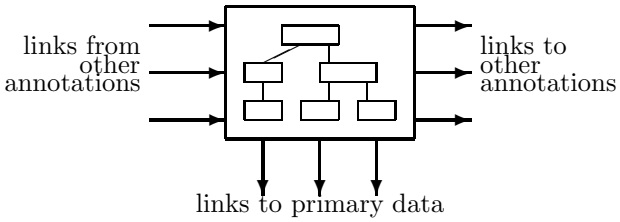


**Figure 1.** A link object, schematically.

discussed here, and are not necessarily traversed using the same mechanism that we will use to traverse the annotation links.

A problem that can arise in a data model that specifies that there can be annotated data fragments, is that there might be two or more references to overlapping data fragments. This implies that when a new annotated substructure is to be declared to the system, all potential relationships between the annotations that refer overlapping fragments should be specified so that this can be deduced by the system. This is an implementation issue investigated also in [12] for the Graphitti project.

The content related to an annotation node is stored in an annotation database. Due to its complex structure, an annotation is defined as an XML object and will be stored in an object-oriented database.

## 3. THE ANNOTATION GRAPH

As we have shown in the previous chapter, annotations refer not only to the primary data, but to each other as well. The presence of inter-annotation links generates additional complexity for the system to manage, and results in a rather complex structure in which three elements are present:

**i)** a graph (a $DAG$ in our model) whose vertices are annotations and whose edges are inter-annotation links;

**ii)** each vertex of the graph (annotation) is not atomic, but is a tree that represents the internal structure of the annotation;

**iii)** from each node of the tree we can "hang" one or more fragments of the primary data through a link represented by a query.

The directed acyclic graphs (DAGs) provide a sufficiently powerful model for the annotation structures we are interested in. DAGs appear in models where it doesn't make sense for a vertex to have a path to itself. They are an extension of hierarchies in which a specialized term (child) can be related to more than one less specialized term (parent). To make a biomedical example, consider the *hexose biosynthetic* process. In a type-of relation, this process has two parents, *hexose metabolic process* and *monosaccharide biosynthetic process*. This is because biosynthetic process is a type of metabolic process and a hexose is a type of monosaccharide. When any gene involved in hexose biosynthetic process is annotated to this term, it is automatically annotated to both hexose metabolic process and monosaccharide biosynthetic process.

The relationships in our data model are modeled using a directed acyclic graph like the one shown in figure 2, in which we also represent the links between primary data that might exist but that, as we mentioned, remain outside of the data model, and can be traversed outside the system using the normal navigation instruments associated with the data.

The edges of the graph can be annotated with the type of relation they represent and a *value* that specifies that relation. Therefore our model is a colored graph with values associated to every edge, that is, the model is a graph $G = (V, E)$, where $V$ is the set of annotations, each one being a tree, and $E \subseteq V \times V \times P \times \mathrm{dom}(P)$,

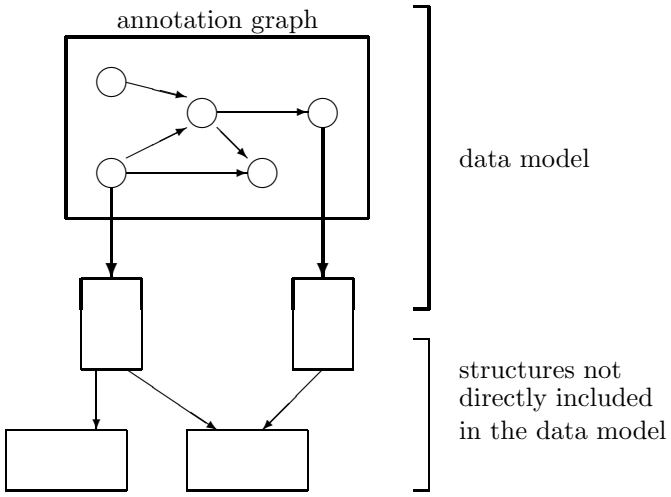


**Figure 2.** The overall data model.

where $P$ is the set of edge types (colors). The semantics of an edge $e = (u, v, t, x) \in E$ is that the edge goes from node $u$ to node $v$, is of type $t$, and has associated a value x drawn from those admissible for that type.

For example, to represent a *part-of* relationship between the annotation part $p$ and the annotation $a$ that includes it we define an edge $(p, a, \text{part-of}, \perp)$. This is translated, for implementation purposes, in an XML structure by creating identifiers that could differentiate between annotation parts, but still show the annotation mother legacy. The nodes can be connected via edges of arbitrary types. We divide $V$ in two disjoint sets: $V = A \cup D$, where $A$ is the set of annotation nodes and $D$ is the set of primary data nodes.

Some examples of edge types with the relative constraints are the following:

**i)** <> denotes an unnamed concept, specifying a simple edge type "references".

**ii)** *annotates* is used to represent edges from annotations to documents. If an annotation is created for a document, a link of this type is defined between them. Since we are interested in fine-grained annotations, e.g., regions in an image or text fragments such a paragraphs, we assume a set of of document context descriptions. Such a description includes information about the document context in which an annotation occurs and is modeled as a property of the relationship. In our current system, we employ two types of descriptions. For text-based documents, we employ a restructured document model. In particular, we employ a transformation mechanism to handle HTML and plain text documents as XML documents. Thus, xpath expressions[11] are used as document context descriptors. For image based data and annotations, we use region, point, and scale information to encode descriptions for fine-grained annotation. Annotates edges can only originate from an annotation, and can end either in an annotation or in a primary datum.

**iii)** *annotatedBy* is the inverse of annotation. So:

$$\forall e \in E.(\text{type}(e) = \text{annotates} \;\Rightarrow\; \text{orig}(e) \in A \wedge\; \exists x.(\text{dest}(e), \text{orig}(e), \text{annotatedBy}, x) \in E) \qquad (1)$$

and

$$\forall e \in E.(\text{type}(e) = \text{annotatedBy} \;\Rightarrow\; \text{dest}(e) \in A \wedge\; \exists x.(\text{dest}(e), \text{orig}(e), \text{annotates}, x) \in E) \qquad (2)$$

**iii)** an edge of type *ofConcept* represents the fact that an annotation is based on a certain concept, i.e. assigns a concept to the annotated document and instantiates the properties defined by the concept.

**iv)** *hasAnnotation* defines the inverse relation of *ofConcept*. Similar relations hold as seen in the previous case.

We stipulate that every annotation must be related to a concept.

## 4. QUERIES ON THE MODEL

There are three different types of query that we can do on the model and that we must take into consideration. First, we can query the primary data directly, issuing http requests, relational queries to relational data bases, and so on. These queries are not properly part of our model, and they will be issued to the data repositories in which the primary data are stored using whatever query instrument these repositories put at our disposal. The annotations that link to these data will contain the specification of the data repository, information on how to issue a query, and the text of the query to be issued. The data will be obtained simply by executing the queries on the corresponding data source.

Second, each annotation is a tree that, in our implementation, is represented by an XML structure. Thus it seems logical to use one of the existing query languages for XML, such as xpath.

Finally, querying the graph structure requires the definition of a new language and the implementation of a query processor and data base for it. The graph query language that we use is, essentially, a modification of regular expressions where node symbols are replaced by xpath queries on the structure of the annotation, and additional conditions can be imposed on the type or value of the edges that join the nodes in a path. A path formula $\phi$ is composed according to the following syntax:

$$
\begin{array}{lll}
\phi & ::= & [\xi] \mid \phi + \phi \mid \phi\phi \mid \phi\{\zeta\}\phi \mid (\phi) \mid \phi^* \\
\xi & ::= & \text{<XP>} \mid \text{t} \\
\zeta & ::= & \text{T:<type>} \mid \text{C:<cond>}
\end{array}
$$

Where XP is an xpath query, <type> is an edge type, and <cond> is a boolean condition on edge values. The expression $[\xi]$ matches every annotation that satisfies the xpath query $\xi$; the semantics of $\phi + \phi'$, $\phi\phi'$, and $\phi^*$ is the usual one for regular expressions, and $\phi\{\zeta\}\phi'$ matches a sequence of two paths $\pi - \pi'$ if $\phi$ matches $\phi$, $\pi'$ matches $\phi'$, and the edge between the two paths satisfies the condition $\zeta$.

The graph path query does not produce results directly. Rather, the results that it returns are all the results produced by the node queries for the nodes that compose a matching path.

This query machinery gives us enough expressive power to express many queries of interest in an annotation system. Graph structural queries such as *Return a datum D annotated with A1, if no other set of annotations pointing to A1 are canceling A1*, or *Return the annotations that point to an annotation A that points to a certain datum D.*

One limitation of the graph query language is that, being based on paths, it can't answer graph structural queries, such as *Find the smallest sub-graph that connects all annotations with the keywords K1 and K2.* This kind of queries can be answered either defining a full graph pattern language (a solution that for the time being seems impracticable due to the extreme complexity of query evaluation and the lack of efficient algorithms), or by including the path language in a framework that allow graph composition operations, such as was done in[12] with the monoid calculus.

## 5. IMPLEMENTATION ISSUES

As part of this work, we developed an annotation program that allows users to create and edit structured annotations and to connect them to data. The editor's visual interface offers a tree-like view of the structure that allows the user to display and manipulate portions of the annotation. As we described in the previous section, an annotation is composed of hierarchically organized parts, and has a tree structure. For storage purposes, the tree is expressed using XML syntax.

When opening an annotation for visualization, the XML is parsed and its tree-like structure is displayed in the left side of the editor window. The tree can be expanded and by clicking his nodes the user can chose the way he wants to visualize the underlying data. By clicking once on a node, the corresponding annotation part will be displayed in the central panel, and by double clicking the node, the whole portion of the annotation rooted at that node will be displayed. Using this view, annotation parts can be created, edited, merged, split, deleted.

An annotation can also be marked *obsolete* while being kept in the data base for future reference. Obsolete annotation can't be edited, but they can be linked to. After editing an annotation or an annotation part, the user has to create the links with the primary data or other annotations. By pressing the Link button, in the right side of the selected annotation part will appear a editing cell that bounds the URI written down to the XML corresponding to the annotation part. The same mechanism is implemented for linking an annotation or just part of it. In order to query the annotations, and to execute the queries that make up the links, we need a

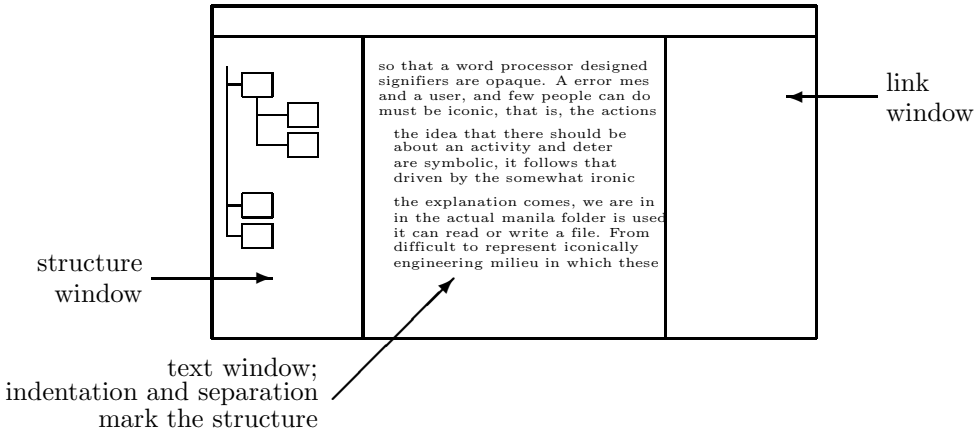


**Figure 3.** The annotation interface

suitable storage and query service. We have decided to use the MonetDB as the database system for storing the XML annotation files.

MonetDB is a open-source database system for high-performance applications in data mining, OLAP, GIS, XML Query, text and multimedia retrieval. MonetDB/XQuery provides XML database functionality comprising quite complete support for the Xquery language, including modules, transaction-safe XQUF updates, user-defined functions, and a query cache (using Xquery modules). The system has very high performance and is scalable to large XML collections. Clients gain access to the server through a internet connection or through its server console. Access through the internet requires a client program at the source, which addresses the default port of a running server. At the server side, each client is represented by a session record with the current status, such as name, file descriptors, namespace, and local stack. Each client session has a dedicated thread of control, which limits the number of concurrent users to the thread management facilities of the underlying operating system. A large client base should be supported using a single server-side client thread, geared at providing a particular service. The mclient program is the universal command-line tool that implements the MAPI protocol for client-server interaction with MonetDB.

Being stored in an XML data base, the data structure is also defined in XML, and its specification is given using the XML schema formalism. However, due to the notorious impenetrability and verbosity of the XML notation, we will express it here using a more standard (and refreshingly simple) programming-style declaration:

```
type annotation is
        begin
            ident: ID;
            element: list of part;
        end;

type part is
        begin
            llst: list of link;
            text: string;
            created: date;
```

```
        modified: date;
        element: list of part;
    end;

type link is locallink or remotelink;

type locallink is target: IDREF;
type remotelink is href: URI;
```

Note that there is a formal reference between an ID (the value of an identifier when it is declared) and an IDREF (the value of an identifier when it is referred). The reason is that an ID is a skolem constant and therefore unique: in the system there can't be two identical ID while, of course, there may be as many copies as necessary of an IDREF.

After editing an annotation, the information introduced by the user is saved as an XML file. As we can see in the schema, there are different types of data corresponding to an annotation, i.e text, dates, links. These sections are saved in the correspondent tags of the XML, for every new created part being also saved an ID that implements a mechanism of part recognition as the proper annotation child.

Querying and navigating an annotation graph is supported by two kinds of operations: selection and path traversal. Path traversal enables following links between nodes of the graph. Given a start node and a relationship, it would to be returned the set of target nodes based existing edges. The query family and the algebra operations are not yet conceptualized. In order to be able to formulate queries involving query operators for path traversal, we will design a simple language in the spirit of XPath. The query engine for this language could be implemented as a basic Web service providing a SOAP interface. The idea is to accept a query expression and to translate it to a relational algebra expression that can be interpreted and derived by the Xquery and then be sent to the back-end DBMS for evaluation. The result set should be encoded into a XML document and then returned to the client. The selection queries are those on the original data, and do not need querying the graph structure. They could consist of http request, for accessing web pages or web documents. This type of queries doesn't involve the definition of a new language, the annotation that annotates this type of data contains the http request that will be parsed and executed from the program. In the case of annotated data that resides on remote databases, the annotation will contain the information about the database server and all the information needed to be accessed through the SQL or Xquery database languages. Momentary, the data within DBMS can obtained by being queried only with the help of SQL or Xquery, XPath query languages. Another type of selection queries is the case of annotation retrieval. If doesn't involve graph traversal, some annotations could be returned directly by the use of a query pattern already defined such as Xquery or XPath. Being defined as XML, the annotation can be retrieved by employing an existing XML Query Language.

Through the techniques implemented, can access data across the network from remote machines, as well as the files that reside locally. Based on the query language model implemented, on HTTP access and well known query services, annotations and data annotated can be concise and easy to deal with. The functions of this tool include highlighting texts, inserting and editing annotations, organizing and presenting annotation parts hierarchically, as well as linking annotations with their referents. The most important functionality is offered by the flexible annotation model which allows a high degree of explicitly in data annotation.

## 6. CONCLUSIONS

In this paper we have presented the general design of the *annotation engine* that we are developing at the Universidad Autónoma de Madrid. Annotations are complex objects, and the unstructured way in which scientists in the field use them contributes to give them a complex structure. An annotation system is an mediation center for what we have called the *primary data* and, at least in the sciences, a tremendous source of information itself, so that annotations are not only an element that agglutinates the primary data by querying them, but are themselves the object of possible queries. All this represents a significant data management challenge.

Our main goal was to create a design suitable for the *post-relational* data world, so to speak. Like many passages from the modern to the post-modern era, the post-relational world entails breaking down of a universal

paradigm (namely, the relational) without replacing it with another but, rather, opening up to the possibility of multiple, conflicting models. In the world of data management this somewhat lyric premiss translate to the mundane observation that data come in a variety of formats and of structures (or lack thereof), with a wide range of querying capabilities (or lack thereof).

We have created a structure flexible enough to deal with a variety of data sources by defining links abstractly as queries, and providing the mechanisms for including these queries as part of the annotation specification. In addition to this, we have considered annotations as structured objects, in particular by giving them a tree structure, so that the connections with the data can be restricted to parts of an annotation.

This system is still work in progress. The regular expression query language still needs some tuning and there are still many efficiency issues related to its implementation. Also, so far our graph queries are limited to paths but, in order to provide a truly flexible, it will be necessary to extend the language to handle sub-graph queries. It is not clear at this time whether the best solution will be to try to create a pattern language of graph structure or to enrich the path language with graph aggregation operations. Future research will hopefully give an answer to these questions.

## REFERENCES


1. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco:Morgan-Kaufmann, 1999.
2. T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology* , pp. 195–7, 1981.
3. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *10th IEEE International Symposium on High Performance Distributed Computing*, pp. 181–4, New York:IEEE Press, 2001.
4. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener, "The lorel language for semistructured data," *International Journal on Digital Libraries* **1**(1), pp. 68–88, 1997.
5. L. Sheng, Z. M. Özsoyoğlu, and G. Özsoyoğlu, "A graph query language and its query processing," in *Proceedings of the 15th international conference on data engineering*, 1999.
6. P. Buneman, M. Fernandez, and D. Suciu, "UnQL: a query language and algebra for esmistructured data based on structural recursion," *The VLDB Journal* **9**, pp. 76–110, 2000.
7. M. Consens and A. Mendelzon, "GraphLog: a visual formalism for real life recursion," in *Proceedings of the nineth SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pp. 404–416, 1990.
8. B. ten Cate, "The expressivity of xpath with transitive closure," in *Proceedings of the SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pp. 328–37, ACM Press, 2006.
9. A. O. Mendelzon and P. T. Wood, "Finding regular simple paths in graph databases," *SIAM Journal on Computing* **24**(6), pp. 1235–58, 1995.
10. D. Srivastava and Y. Velegrakis, "Intensional associations between data and metadata," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 401–12, ACM Press, 2007.
11. J. Clark and S. DeRose, "Xml path language (xpath)," recommendation, World Wide Web Consortium, on-line:1999.
12. S. Santini and A. Gupta, "A calculus and algebra for querying directed acyclic graphs," in *XV Journadas de Ingeniería del software y bases de datos*, J. Riquelme and P. Botella, eds., Barcelona:CIMNE, 2006.