Object Database Support for Digital Libraries

Serge Abiteboul*

Stanford U. and INRIA-Rocquencourt

Abstract. In this paper, we discuss some aspects of database support for digital libraries.
From a DL perspective, database systems, and in particular, object database systems provide a nice basis for future DL systems. More generally, database research provides solutions to many DL issues even if these are partial or fragmented. When possible, work should not be duplicated and good software and ideas should be reused.
From a DB perspective, we want to stress that digital libraries propose beautiful applications and challenges to DBMS technology. They suggest a number of improvements to DBMSs that could be beneficial beyond DL applications.

1 Introduction

The success of the Internet and the Web, and the popularity of Intranet information servers have brought Digital Libraries to the forefront of research. Digital libraries (DL) until recently have focused on issues such as data digitization, search in document collections, and relevance ranking. The scaling up of DL applications stresses new issues such as the organization and access of distributed massive volume of documents (e.g., Web). Existing systems meet difficulties in scaling up and in adapting to application demands (e.g., access control, large volumes, distribution, payment). In this paper, we consider how database technology can be used to support DL applications systems. We will see that many concerns in DL are also found in databases and that if an (object) database approach does not solve all the problems, it provides an interesting angle to attack many.

An essential task of digital libraries is the management of information, i.e., data. Not surprisingly, many features of database systems seem desirable in a DL context: secondary storage management, persistence, transactions, concurrency control, recovery, versions, etc. Furthermore, even if DL access languages are primarily based on information retrieval techniques (pattern matching), one can argue that logic-based database query languages (in the style of SQL) appropriately extended with DL specific features should form a sound basis for DL

* Email: Serge.Abiteboul@inria.fr. Work supported in part by CESDIS, NASA Goddard Space Flight Center; by the Air Force Wright Laboratory Aeronautical Systems Center under ARPA Contract F33615-93-1-1339, and by the Air Force Rome Laboratories under ARPA Contract F30602-95-C-0119.

query languages (even if this means introducing fuzzy logic aspects). In view of all this, one may wonder why database systems are not systematically components of DL systems. A main reason is that DL mainly focused on accessing data (not modifying it besides appends) whereas a lot of database technology concerns the management of data changes. As a consequence, databases were viewed as too complex for DL applications. Indeed, for small scale data collections, a file-based approach is easier to install. However, as the collection grows, when collections are integrated, or when new functionalities are added, a database solution becomes compulsory.

Unfortunately, switching to a DBMS does not solve all the problems. Database management systems (DBMSs) were originally designed for business applications. The most successful ones, the relational systems [Ull89], deal with relations, i.e., sets of fixed records. In a DL environment, the information consists primarily of loosely structured data including text, image, graphics and sound, which is hard to fit into two dimensional tables. A second difficulty is that relational systems were meant to be strong, protected, isolated boxes. In contrast, DL systems often stress openness to other system components (e.g., graphics libraries) and in particular to the network. For these reasons, we will argue that the new generation of database systems, *object database systems* (and to a lesser extent *object-relational systems*) provide a sound basis for DL applications.

Object database systems [Cat94] emerged in the late eighties [ABD+89] as the merging of object technology (e.g., C++, Smalltalk) and relational technology (e.g., Oracle, Sybase). Object databases provide persistence, secondary storage management, concurrency control, recovery, etc. They also allow programming database applications using object-oriented languages (primarily C++). This simplifies enormously the task of application programmers since they manipulate the same concept (an object) from the disk (persistent storage) to the processor (inside an application) to the screen (user interfaces). The nineties saw the development of OODB systems (Object Store, O_2) and of a standard by the *Object Database Management Group* (ODMG) that regrouped the main OODB vendors. In particular, the *Object Query Language* (OQL) was adopted as a standard.

Besides the increase in programmer's productivity, the advantages (in general and for DL in particular) of OODB are (i) the ability to represent complex data, and (ii) more openness to the external world (e.g., Web, Java, CORBA, language bindings) than relational systems. One should also mention that relational vendors somewhat followed the object trend by introducing object features in their systems (e.g., last release of Oracle) leading to what is sometimes called *Object-Relational* systems. We believe that object databases offer cleaner solutions and somewhat better master the object-oriented dimension. For instance, OQL is a clean and implemented language whereas we see SQL3 as some very complicated monster with an unclear semantics and no implementation so far. We will ignore the object relational systems here.

In this paper, we mention some issues that are raised by DL applications and the extensions to database systems that they suggest. The theme of the paper is guided by some projects in which I participated at INRIA and at Stanford during the last few years. (See www-rocq.inria.fr/verso.) These were projects on document or hypertext databases, cartographical databases, and object repositories for software engineering or for the Web. All these projects dealt with data that was not as regular as in conventional (relational) database applications. In them all, interoperability with external software was crucial. The same can be said for digital library applications in general. We would like to warn the reader at this point that this is by no means an attempt to cover the domains of databases and digital libraries, not even their intersection. Such an endeavor would have required more time than we had to prepare this invited lecture. Essential problems, e.g., data fetching and caching for video servers, that are fascinating database issues in the context of digital libraries, are beyond my personal expertise and will be ignored.

The paper is organized as follow. In Section 2, we discuss the particularities of DL data compared to data in more conventional database applications. In Section 3, we consider the issues of model and languages for such data. Data distribution and integration form the topic of Section 4. Section 5 considers the enrichment of a collection of documents with a semantic layer of more structured information. Section 6 deals with the management of changes. In Section 7, we briefly mention issues such as payment and electronic commerce.

2 Semistructured Data

What kind of data are digital libraries dealing with? Consider for instance data in a museum library. It consists of a collection of images, videos and some text documents. If we consider each piece of data (image, video, page) as a separate entity, we may be missing the point. For instance, a museum curator may want to organize a virtual visit of the museum collections around a particular theme. So, relationships between documents, and furthermore, with documents in *external* sources, are essential to use the library in a meaningful way. Also, a key problem is the automatic extraction of such relationships using a variety of tools: domain specific thesauri, ontologies, parsers, text or image analysis, data mining tools, etc.

So, a DL should be thought of as a collection of interrelated objects together with some software to access data as well as to enrich it by discovering and adding semantics to it. In this section, we focus on the data.

Because of external data, and for historical reasons (legacy data), we must live with data that reside in many different forms, ranging from unstructured (plain text, images) in file systems to highly structured (relational data). Data is accessible through a variety of interfaces including Web browsers, database query languages, application-specific interfaces, and data exchange formats. We use the term *semistructured data* for this data that is neither raw data (e.g., images) nor strictly typed data (i.e., not table-oriented as in a relational model or sorted-graph as in object databases). Examples of semistructured data are SGML (or HTML) files or more generally data exchange formats (e.g. ASN.1), hypertexts and programs. Semistructured information also typically arises when integrating several (possibly structured) heterogeneous sources.

Let us try to isolate now some aspects that often characterize the semistructured data found in DL and contrast it with data found in more standard DB applications:

1. **The structure is irregular:** Different types may be used for the same kind of information, e.g., prices may be in dollars in portions of the database and in francs in others. The same piece of information. e.g., an address, may be structured in some places as a string and in others as a tuple.
2. **The structure is implicit:** For instance, documents consist often of a text and a grammar (e.g., a DTD in SGML). The parsing of the document then allows one to isolate pieces of information and detect relationships between them. Some computation is required to extract this structure that may be costly (in particular, if the document is in plain text).
3. **The structure is partial:** To completely structure the data often remains an elusive goal. Parts of the data may lack structure (e.g., bitmaps); other parts may only unveil some very sketchy structure (e.g., plain text).
4. **The structure is indicative versus constraining:** In standard DB applications, a strict typing policy is enforced to protect data. In many DL applications, a strict typing policy often is infeasible or would be viewed as too constraining.
5. **A priori schema vs. a posteriori:** Traditional database systems are based on the hypothesis of a fixed schema that has to be defined prior to populating the database. This is often not the case in DL applications where the notion of schema is often posterior to the existence of data.
6. **The schema is very large and often ignored:** As a consequence of the heterogeneity of sources and of integration, a precise schema would typically be quite large. The usefulness of this "typing" becomes unclear. Indeed, queries, that have a more discovery nature, tend to ignore it. In general, the importance of schema is toned down and the separation between schema and data is blurred (e.g., "self-describing data").
7. **The schema is rapidly evolving:** In standard database systems, the schema is viewed as almost immutable, schema updates as rare, and it is well-accepted that schema updates are very expensive. In contrast, many DL applications (e.g., for genome data) expect the schema to change quite rapidly, at the same speed as experimental techniques are improved or novel techniques introduced.
8. **The type of data elements is eclectic:** The structure of a data element may depend on the user viewpoint or on the particular tool that is using it. Also, this type may vary in time. For instance, an object may be first a file, become a BibTeX file after classification, obtain an *owner*, *creation-date*, etc., after some information extraction phase, and finally, turn into a collection of reference objects after parsing. So, the type of a piece of information has to be more eclectic than, say in standard database systems.

It should be noted that although we attempted to characterize DL data with these various aspects, a particular DL application may not satisfy all. For instance, an organization may impose some very strict a priori typing on all internal documents. Also, the natural trend towards imposing some standards for DL systems will tend to favor a priori typing. However, a combination of some of these aspects will often characterize DL data and distinguish such data from data found in more standard database applications.

Let us now consider the issue of storing DL data in object databases. The richer structure facilitates the task compared to for instance storing the data in a relational system. However, some issues are raised. We need to develop models and query languages for it. We will see the need for new query optimization techniques, and in particular, the need to integrate optimization techniques from various fields (e.g., database indexes and full-text indexes). Some standard database system issues such as transaction management, concurrency control and error recovery have to be reconsidered, in particular, because the notion of "data item" becomes less clear: the same piece of data may have several representations in various parts of the system, some atomic, some complex. Physical design (in particular clustering) is seriously altered in this context. Finally, it should be observed that, by nature, a lot of the data will reside outside the database. The optimization of external data access (in particular, the efficient and selective loading of file data) and the interoperability with other systems are therefore key issues.

3 Modeling and languages

The relational model is too rigid to provide the desired basis. On the other hand, the standard data model for object databases, the ODMG [Cat94] data model, provides the right primitives, an appropriate framework for extensibility, and the desired flexibility. The ODMG model supports the notions of objects organized in classes with a class hierarchy, the standard atomic types, and *set, list, bag, array* as structuring constructs. All this seems necessary and useful to describe the kind of complex data found in DL applications. In particular, the notions of object and classes are essential for handling multimedia types such as image, sound, or video.

The following extensions were suggested by some previous work to manage SGML documents in an O_2 database system [ACC$^+$97]:

1. Null values: these are given lip service in the relational and the ODMG models and more is needed here.
2. Heterogeneous collections: collections need often to be heterogeneous in this setting. So, there is the need for some union types.
3. Text with references: text is essential in the DL context. Two important issues are (i) references to portions of a text (references and citations in LaTeX), and (ii) external references within a text (HTML links).

The strict typing of ODMG is useful when the structure is precisely known. In some cases (e.g., for loosely structured data), this strict typing may become a handicap. Two similar models have been independently proposed to handle such situations [AQM+97, BDHS96]. In these models, data consist of labelled graphs. Entities are represented by *objects* (vertexes of the graph) and relationships by edges. Some objects are atomic and contain a value from one of the disjoint basic atomic types, e.g., `integer`, `real`, `string`, `gif`, `html`, `audio`, `java`, etc. All other objects are complex. Their value (the outgoing edges) can be viewed as a set of *object references*, a set of $(label, oid)$ pairs. The labels are taken from the atomic type `string`. Figure 1 provides an example of some "OEM" (for *object exchange model*) data.

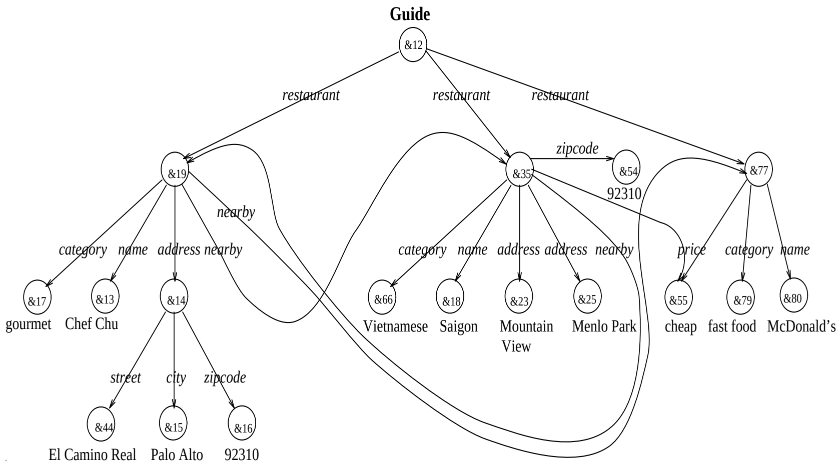

Fig. 1. Some OEM data

The interesting point is that this kind of information can be naturally embedded in an extension of the ODMG model. Indeed, Lorel [AQM+97] is an extension of the ODMG query language (OQL) that allows accessing ODMG or semistructured data, or a mixture of both. Lorel has been implemented on a Stanford object database server. A second implementation on top of the O_2 system allows mixing ODMG and semistructured data.

In the context of semistructured data, the query language has to be more flexible than in conventional database systems. Typing should be more liberal since by nature data is less regular and the user is less aware of the structure even when it exists. What should we expect from a query language?

1. searching for pattern in an information retrieval style [Rie79];
2. standard logical database-style query primitives;
3. navigation in the style of hypertext or Web browsing;
4. querying both the data and the schema in the same query as in [KL89].

Also, the language should have sound theoretical foundations, possibly a logic in the style of relational calculus. (There has been a lot of work on calculi for object-based models, but in general, under the assumption of rigid typing.)

Besides some syntactic sugaring, Lorel extends OQL in two essential ways: (i) generalized path expressions that allow some form of pattern-based navigation in the graph and (ii) an extensive use of coercion to relieve users from the mundane tasks of specifying types or using explicit type coercion. We will not attempt to describe the language here. But to illustrate the previous discussion, consider the following Lorel query on a car rental library:

```
select A, M, M.report
from rental-agent A, A.car C, C.model M
where  M.report.rating > 5
  and  A.location near ''Disneyland''
```

The query requests, for each agent located near Disneyland and providing cars of models with rating higher than 5, the list of all such models and the reports for these models.

The casual DL user should not compare the previous query to queries of the form "car and (rating $>$ 5) and Disneyland" that he/she would be more accustomed with. It is very unlikely that a user would ever have to type the previous Lorel query. As is the case with SQL, Lorel queries would most likely be automatically generated by an application. Such queries on the other hand facilitate the work of application programmers and also the dialog between (distributed) system components.

The above Lorel query is also an OQL query (with the exception of *near* that could be handled by a method). This is a correct OQL query if all data is stored in an ODMG database. However, suppose that reports are in fact HTML documents and that a *near* method is implemented by calling an external source of geographical informations. The same query is a Lorel query that would also provide the desired result in such a case. Note that the evaluation of the query requires the cooperation of many tools. Observe also that the rating could be stored in reports as integers, strings, reals or other compatible types, therefore the need for automatic coercion.

There has been a lot of work on query languages that may be relevant here, e.g., besides the ones we already cited, deductive languages (see [Ull89, AHV94]), query languages for hypertext structures ([MW95, BK90, CM89, MW93]), languages for data integration [PGMW95, PGMU96], and, recently, query languages for the Web ([KS95, MMM96, LSS96]).

Remark. Database query languages have been traditionally used for *extracting* data from a database. They also serve to specify *views*. The notion of view is particularly important here since we often want to consider the same object from various perspectives or with various precisions in its structure. We need to specify complex restructuring operations. The view technology developed for object databases was in part developed to serve such goals, e.g., [dSAD94] as well as the restructuring mechanisms proposed in UnQL [BDHS96].

4 Data distribution and integration

The growth of digital libraries probably depends as much on the integration of existing libraries as on the scaling-up of individual ones. Data integration [SL90, LMR90, Bre90] is an old topic in databases. One can distinguish two main approaches:

1. data warehouse: data from many sources is transformed, cleansed, materialized in a database, and a unique interface to it provided. (See, e.g., [Wid95].)
2. mediator architecture: the data resides virtually at a *mediator*. When a query arrives it is broken into multiple queries to the sources. The results are combined to form the result to the original query. (E.g., see the TSIMMIS project [QRS+95].)

In both cases, if the sources are heterogeneous, *wrappers* have to be introduced. A wrapper provides query translation between the source it wraps and a mediator, thereby allowing the mediator to use a single model and to speak a single language. In the digital library context, the same two approaches can be followed. Novel issues arise, e.g.:

1. efficient maintenance: many DL sources (e.g., genome data banks) are file-based and often provide insufficient support for incremental changes.
2. information retrieval issues: how can we combine rankings from different sources based on different scales? How do we manage the fact that different sources may use different terminologies?

An essential component for being able to interoperate across various digital libraries is the translation from one world to another, e.g., format translation. To conclude this brief discussion on integration and distribution, we would like to mention another issue, the communications between the various components of the system. Distributed object technology provides a sound technical basis. For instance, the Stanford Digital Library Project uses CORBA (the distributed-object standard developed by the Object Management Group) to implement information access and payment protocols across a wide variety of DL sources [PCGM+96]. In that respect also, the use of an object database as a repository for a digital library would facilitate development because of the existence and efficiency of CORBA bindings. (Although it is possible to interface a relational database with CORBA, it is much more cumbersome to do so, and the required translation back and forth to the relational world comes at some cost in performance.)

5 Structuring data

When data collections grow (and they tend to grow naturally with more and more data becoming available electronically), it does not always translate into better quality or more usable data. A typical example is the Web where search queries often result in thousands of answers that tend to drown the user in data. Indeed, it is often preferable to only see a more organized layer of information that enriches the large data collection by filtering, removing duplicates, organizing it by topic, providing road maps, etc.

Starting with data with little explicit structure, we need to extract structuring information and organize the data to improve performance and to facilitate logical access to this data. To consider a simple example, suppose we have a number of files containing bibliography references in BibTeX and other formats. We may want to extract (in a data warehousing style) the titles of the papers, lists of authors and keywords, i.e., the most frequently accessed data that can be found in every format for references, and store them in an object database. Note that this extraction phase may be difficult if some files are structured according to formats ignored by our system. Also, issues such as duplicate elimination have to be faced.

In general, we need tools to extract information from files including classifiers, parsers, and also software to extract cross references (e.g., within a set of HTML documents), information retrieval packages to obtain statistics of word occurrences (or groups of words) and statistics for relevance ranking and relevance feedback. More generally, one could envision the use of general purpose data mining tools to extract structuring information.

The object database then provides a structured layer above the layer of more unformed data with links to pieces of information in the files, so that all data remains accessible. What exactly do we achieve using such a (more or less loose) structuring layer?

1. We have enriched our data with more semantics by classifying data pieces and establishing relationships between them. For instance, we may have populated the domain with semantically meaningful entities (persons, addresses, companies, bibliography references, experiments, genes, etc.) and established relationships between these entities.
2. We facilitate query formulation by providing more precise *form-based* queries (in the QBE-like interface style) and we have set the ground for *cooperative answers*. E.g., instead of simply returning an empty or meaningless answer, the system may explain why the answer is empty and propose alternatives.
3. This structured layer coupled with standard database query optimization techniques can provide important performance gains in answering the most common queries since it can be seen as some form of view materialization. It can also be seen as additional access structures (beyond full-text indexing) that accelerate the evaluation of queries.
4. Finally, this structured layer is also a good information source for explaining the content of the library and the kind of information one should expect to find in it. It may also include meta-information such as the quality of the data the library contains.

Such a layering architecture is called *structured maps* in [DMRA97].

6 Versioning and the temporal dimension

We believe that actual DL systems are too weak with respect to change control and more generally with respect to the handling of the evolution of information in time. Existing database systems provide some solutions. For instance, they provide versioning mechanisms. Also, there has been quite a lot of work on *temporal databases* [SA86, Soo91] and temporal query languages.

Unfortunately, a difficulty when trying to adopt techniques from temporal databases in a DL context is that most of the work in temporal databases is based on the relational model. A change in the relational context is simply the insertion, deletion, modification of some tuples. In the DL context, this is not so easy. For instance, there may be changes *within* a text. If this text is organized hierarchically, changes have to be considered with various granularities (e.g., paragraph, subsection, section). Finally, when we move to a graph of objects (e.g., in HTML), then we obtain new kinds of atomic changes (e.g., adding an edge to the graph or moving an entire subtree). Thus, there is first a problem of representing the changes. This is a problem encountered in object databases as well. This should not come as a surprise since object databases provide some of the structuring mechanisms naturally found in hypertext data.

In [CAW97], this representation problem is studied. An extension of Lorel is proposed to handle changes. A motivation is that we are often as interested in the changes as in the data itself. For instance, consider the *Palo Alto Weekly*, a local newspaper that maintains a Web site [Gul97] providing information about restaurants in the Bay Area. Most of the data in the restaurant guide is relatively static. But as often the case in DL applications, we may have a particular interest in the dynamic part of the data, in the changes. For example, to find all restaurant entries that were added after January 4th, 1997, we can use the following Chorel (the extension of Lorel) query:

```
select guide.<add at T>restaurant
where  T > 4Jan1997
```

Intuitively, the temporal information is conceptually added to the graph data using annotations (add, remove for edges; create, update for vertexes), and patterns of these annotations are used in queries.

We believe that the support of some form of temporal queries would be useful for many DL applications. For instance, Chorel is used in a *query subscription system* that allows users to send subscriptions such as: "Notify me whenever a new book by Isabel Allende comes out", or "Notify me weekly on every new publication on electronic commerce with Java in the title".

7 Transactions and Electronic Commerce

Digital libraries are facing a new array of problems when payment is considered. This is leading to the field of *electronic commerce* [YAF95], intricately connected to that of digital libraries. Here, we consider briefly database support for electronic commerce as well.

A number of database features are useful in an electronic commerce context. Primarily, product modelling can benefit from a database approach. Furthermore, in a commercial transaction, participants staking their financial well-beings will be reluctant to accept a system that they cannot trust or cannot gain a precise behavioral understanding of. To answer this issues, a database approach presents the following advantages:

1. Database systems provide concurrency control and recovery mechanisms that provide some guarantee of correctness. The persistence of data is also essential to be able to control a business transaction or to verify it a posteriori.
2. Also, we believe that electronic commerce should, like databases, be developed on some formal (logical) foundations and prefer logical languages (SQL-style) to large chunks of unverifiable code.

One essential component of electronic commerce does not fit in standard database technology: the process modeling component of business transactions. However, *active databases* [WC96] that augment databases with some form of rules in the style of production rule systems do capture this component. Indeed, we use in [FAY97] an active database approach to model business transactions. A formal semantics of the model (again to provide a precise understanding, that we believe is crucial here, of the system) is obtained using Gurevich Abstract State Machines as underlying formalism [Gur94].

To conclude this section, let us revisit somewhat more precisely the notion of transaction in this context. Transactions are first used at the database level to guarantee the correctness of storage. The transaction atomicity allows to guarantee data consistency as in standard databases. In a distributed context, transactions can be used to provide consistency at a more global level, e.g., guarantee that a relation is properly replicated in two sites. Now a *business transaction* may actually span a long period of time. The standard technologies for maintaining database transactions (typically 2-phase-locking or timestamping) are not appropriate and long transactions have also to be supported.

In this context again, object databases do provide a better basis than relational systems by facilitating the interoperability with other software components. In particular, we believe that Java will play a major role in enabling EC, and object databases provide bindings to Java that do not necessitate heavy translations from objects to tuples and back.

To conclude this section, we would like to observe that although standard database technology does not provide all the answers (e.g., it lacks support for long transactions and is often too limited with respect to active features), basic mechanisms for (short) transaction support, concurrency control, replication or recovery are essential building blocks for electronic commerce applications.

Acknowledgement We want to thank Sudarshan Chawathe and Yannis Papakonstantinou for comments on a draft of this paper.

References

[ABD+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Proc. of Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 40–57, 1989.

[ACC+97] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and Jerome Simeon. Querying documents in object databases. *International Journal on Digital Libraries*, 1, 1997.

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading-Massachusetts, 1994.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1, 1997.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 505–516, 1996.

[BK90] C. Beeri and Y. Kornatski. A logical query language for hypertext systems. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1990.

[Bre90] Y. Breitbart. Multidatabase interoperability. *Sigmod Record*, 19(3), 1990.

[Cat94] R.G.G Cattell. *The Object Database Standard: ODMG-93.* Morgan Kaufmann Publishers, San Francisco, CA, 1994. with contributions from Tom Atwood et.al.

[CAW97] S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data, 1997. Database group report, Stanford.

[CM89] Mariano P. Consens and Alberto O. Mendelzon. Expressing structural hypertext queries in graphlog. In *Proc. 2nd. ACM Conference on Hypertext*, pages 269–292, Pittsburgh, November 1989.

[DMRA97] L.M.L. Delcambre, D. Maier, R. Reddy, and L. Anderson. Structured maps: Modelling explicit semantics over a universe of information. *International Journal on Digital Libraries*, 1, 1997.

[dSAD94] C. Souza dos Santos, S. Abiteboul, and C. Delobel. Virtual schemas and bases. In *Intern. Conference on Extending Database Technology*, Cambridge, 1994.

[FAY97] B. Fordham, S. Abiteboul, and Y. Yesha. Evolving databases: An application to electronic commerce. In *International Database Engineering and Applications Symposium (IDEAS)*, Montreal, 1997.

[Gul97] P. Gullixson. The Palo Alto Weekly online edition. Embarcadero Publishing Company, Palo Alto, California, 1997. Available at `http://www.service.com/PAW/`.

[Gur94] Y. Gurevich. *Evolving Algebras 1993: Lipari Guide, Specification and Validation Methods.* E. Börger, Oxford University Press, 1994.

[KL89] M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1989.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 54–65, 1995.

[LMR90] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *Computing Surveys*, 22(3), 1990.

[LSS96] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A declarative language for querying and restructuring the Web. In *Proc. of 6th. International Workshop on Research Issues in Data Engineering, RIDE '96*, New Orleans, 1996.

[MMM96] A. Mendelzohn, G. A. Mihaila, and T. Milo. Querying the World Wide Web. In *Proc. PDIS*, 1996.

[MW93] T. Minohara and R. Watanabe. Queries on structure in hypertext. In *Foundations of Data Organization and Algorithms, FODO '93*, pages 394–411. Springer, 1993.

[MW95] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comp.*, 24(6), 1995.

[PCGM+96] A. Paepcke, S.B. Cousins, H. Garcia-Molina, S.W. Hassan, S.K. Ketchpel, M. Rscheisen, and T. Winograd. Towards interoperability in digital libraries: Overview and selected highlights of the stanford digital library project. *IEEE Computer Magazine*, 29(5), 1996.

[PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *International Conference on Data Engineering*, pages 132–141, 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Data Engineering*, Taipei, Taiwan, 1995.

[QRS+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. Technical report, Stanford University, December 1995. Available by anonymous ftp from `db.stanford.edu`.

[Rie79] C.J. Van Riejsbergen. *Information retrieval.* Butterworths, London, 1979.

[SA86] R. Snodgrass and I. Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, September 1986.

[SL90] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *Computing Surveys*, 22(3), 1990.

[Soo91] M.D. Soo. Bibliography on temporal databases. *SIGMOD Record*, 20(1):14–24, March 1991.

[Ull89] J.D. Ullman. *Principles of Database and Knowledge Base Systems, Volume I,II.* Computer Science Press, 1989.

[WC96] J. Widom and S. Ceri. *Active database systems: Triggers and rules for advanced database processing.* Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[Wid95] J. Widom. Research problems in data warehousing. In *4th Int'l Conference on Information and Knowledge Management*, 1995.

[YAF95] Y. Yesha, N. Adam, and B. Fordham. *Digital Libraries and Electronic Commerce,* in *Digital Libraries: Current Issues.* Springer-Verlag, 1995.

This article was processed using the LaTeX macro package with LLNCS style