A Native Extensible XML Query Processor Towards Efficient and Effective MPEG-7 Querying

Federica Mandreoli, Riccardo Martoglia, Mattia Righini and Paolo Tiberio

Università di Modena e Reggio Emilia, Italy
{fmandreoli,rmartoglia,mrighini,ptiberio}@unimo.it

***Abstract*— In recent years the production of massive amounts of visual information has led to the arrival of very large multimedia Digital Libraries (DLs). The key to support efficient search and management operations in such repositories is to exploit metadata information for digital media, such as MPEG-7 [4] based ones, which seem to be the most widely accepted. The underlying XML syntax, together with the high versatility of the provided constructs, make it easy to specify significant and complex queries, however executing them efficiently on huge quantities of data is not a trivial task. In this paper we provide an overview of the XSiter system, a native and extensible XML query processor providing very high performance in general XML querying settings and whose flexible architecture can be easily enhanced to better support the peculiarities of retrieving multimedia objects through MPEG-7 annotation metadata. Further, we consider possible "use-cases" and tasks related to multimedia and video DLs querying and management which our system can successfully accomplish.**

I. Introduction

In recent years the production of massive amounts of visual information has led to the arrival of very large multimedia Digital Libraries (DLs). The key to support efficient search and management operations in such repositories is to exploit metadata information for digital media. In this respect, a multitude of standard initiatives and efforts have been recently proposed. Among these initiatives, the Multimedia Content Description Interface (also known as MPEG-7 [4]) seems to be the most widely accepted by the major broadcasting companies, hi-tech commodity producers and telecommunication service providers that have joined their effort from 1996 to 2001 to develop a metadata standard for multimedia content. MPEG-7 provides a complete set of tools for a detailed description of audiovisual information. Essentially MPEG-7 descriptors are XML documents valid for schema definitions written in MPEG-7 DDL (Data Definition Language). The underlying XML syntax, together with the high versatility of the provided constructs, make it easy to specify significant and complex queries, however executing them efficiently on huge multimedia DLs data is not a trivial task.

Let us first briefly analyze which requirements have to be satisfied by a system that has to efficiently deal with XML documents. Performing query processing on XML data requires to manage the following issues:

- how to efficiently store XML documents;
- how to express queries;
- how to efficiently process queries.

To this end, XML enhanced RDBMSs could be exploited, trying to take advantage of years of research and development on relation databases. However, standard RDBMS supporting XML extensions are not a viable solution: It has been proved that due to the high nesting that characterizes XML documents and due to the typical semi-structured nature of documents, relational technology alone has poor performance. On the other hand, native XML Databases (or XML base management systems [3]) appear today as a very promising new technology, particularly when exploited in scenarios demanding high querying performance. Further, in order to effectively and efficiently manage MPEG-7 documents, a system needs also to [6]:

- manage typed descriptors;
- provide indexing for typed values, text and structure;
- provide specific querying facilities for multimedia applications.

Since MPEG-7 is intrinsically extensible and since it could be used for a wide range of applications, it is essential that systems are also extremely extensible with respect to the features mentioned above.

In this paper we introduce the XSiter (*XML SIgnaTure Enhanced Retrieval*) system, a native and extensible XML query processor providing very high querying performance in general XML querying settings. The paper is organized as follows:

- In Section II we describe XSiter flexible architecture, which exploits specialized algorithms, indexes and storing facilities;
- Then, we show how it can be easily enhanced to even better support the peculiarities of MPEG-7 data and multimedia retrieval (Section III);
- Finally (Section IV) we consider possible "use-cases" and tasks related to multimedia and video DLs querying and management which our system can successfully accomplish.

II. XSiter, an extensible XML query processor

In Figure 1 we can see the abstract architecture of XSiter. The system is essentially composed by three subsystems, that respectively manage, from the top to the bottom, the interaction with the user (*GUI*), the import process of documents and queries and the query processing (*Core System*) and finally

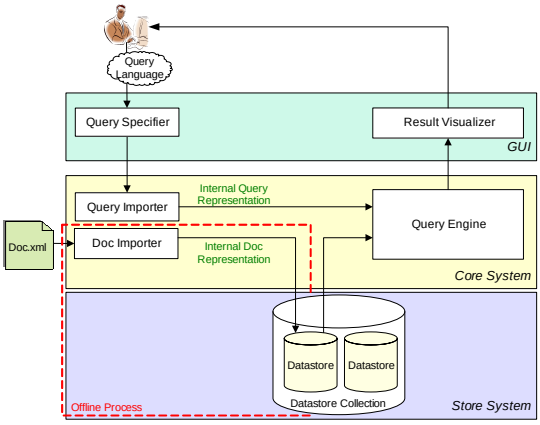


Fig. 1. Abstract Architecture of XSiter

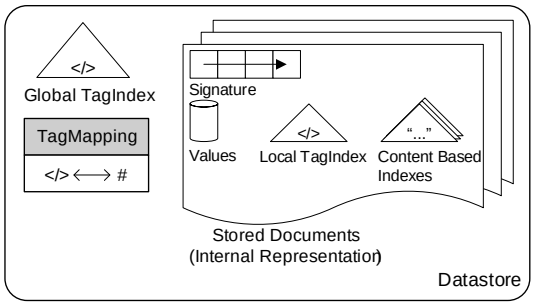


Fig. 2. Structure and content of a Datastore

the persistence of managed documents (*Store System*). Due to the lack of space we will only give an overview of some of the system features, however Figure 1 should be enough self-describing to understand the basic role of each component.

In the Store System, *datastore* structures (Figure 2) are managed. Essentially, a datastore is a collection used for aggregating conceptually related documents and for keeping their representations persistent among different query sessions. Indeed, in order to be processed by the query engine, documents and queries are transformed in an *internal representation*. Such representation is the key of the system efficiency and is exploited by our ad-hoc query processing algorithms. Queries and documents are transformed in an almost homogenous representation; in the following we will briefly describe only the document one, being it the most complex. The chosen internal representation addresses the main issues needed for querying XML documents, which basically consist in finding portions of documents that obey to structural and content constraints, and consists of four main parts (see right part of Figure 2):

- We have a compact representation (*Signature*) based on a numbering schema that is used for solving tree pattern matching (*structural constraints*);
- A simple document summary (*Local Tag Index*) is used as a *filter* for limiting the search space. In particular, for each tag that is present in a document, the first and the last document positions are kept;
- The compact representation does not include *values*, that are stored separately and are evaluated only by need (*value constraints*);
- Finally, values, elements contents or attribute values, can be indexed (*Content Based Indexes*) in order to speed up the search process. Such part is optional and is generated according to user needs.

Finally, along with the document internal representations, in a datastore two shared global structures are also kept (see left part of Figure 2), named Global TagIndex and TagMapping. The Global TagIndex keeps track of which tag is present in each managed document and is used to quickly filter out documents that can not contain matches for a particular query. Instead, TagMapping provides mappings between textual tags and correspondent unique numbers (ids), which are used to store tags in the most compact possible way.

Let us now analyze some of the system features more in detail. As document signature we exploit the *tree signature* structure [7], allowing us to maintain a small but sufficient representation of the tree structures able to decide and solve the structural constraints of the queries. As a coding schema, the *pre-order* and *post-order* ranks [2] are used, enabling us to easily test and discover the different structural relationships between the elements, such as the ancestor, descendant, preceding and following ones.

By exploiting tree signatures properties, and in particular their sequential nature, we have been able to devise and exploit a series of ad-hoc algorithms with optimal I/O performance, as the ones presented in [8]. XSiter provides different algorithms solving the different problems of XML pattern matching: One involving paths and two involving trees. *Path matching* is the most simple type of pattern matching, where the query is expressed as a series of elements disposed in a simple linear structure. More complex queries are ordered labelled trees (*ordered tree pattern matching*), where the order between the elements is important, and unordered labelled trees (*unordered tree pattern matching*). Supporting all the different kinds of pattern matching is essential, particularly for multimedia querying; indeed, as we will specifically see with some examples in Section IV, while there are, for instance, certainly situations where the ordered tree pattern matching perfectly reflects the information needs of users, there are many other that would prefer to consider query trees as unordered.

Besides structural constraints support, XSiter fully supports query evaluation with value constraints, which is a requirement of the utmost importance. Values database simply keeps for each element/attribute the correspondent value. On the other hand, values can also be organized in Content Based Indexes in order to speed up the retrieval process. A Content Based Index is constructed upon a specific tag for a specific kind of search (for example indexes for exact or partial match) and its abstract interface enables search algorithms to retrieve through it all the elements whose content satisfies the specificated condition. Although Content Based Indexes are not mandatory,

it is highly recommended to build them for the elements or attributes that are more frequently queried. The system includes efficient implementations of the most effective index structures available for *exact match* searching, such as inverted indexes for textual data. Standard versions of the matching algorithms provide value support even when specific index data on the queried elements is not available. On the other hand, the system also provides *value-specialized* versions of the different algorithms, taking full advantage of the possibly available content-based indexes built on the content of the document nodes. They enhance standard algorithms in two ways: They recognize more useless elements and are able to avoid the scanning of regions that do not contain any valid match. This further enhances the system performances for solving querying involving values.

In general, all XSiter algorithms efficiently process the supporting data structures in a sequential way, skipping areas of obviously no query answers, whenever possible. The key to their efficiency is to skip as much of the underlying data as possible, and at the same time never return back in the processed sequence. Repositories involving a large number of documents are efficiently managed and queried, also thanks to the document filters exploiting Global TagIndex Data. Further, a filter based on Local TagIndexes is also available, which is used to limit the parts of a document that have to be scanned in order to solve a particular query. These features, together with the minimal memory usage of the algorithms, make the system suitable for querying and managing very large documents.

XSiter is currently implemented as a general purpose system, meaning that little special domain optimizations have been currently applied but the architecture was developed to be simply extensible. In particular, current algorithms use a very high abstraction level of content index access that enables us to substantially use different kinds of indexes without changing the search algorithms. Further, specialized index structures can be easily integrated and exploited in our system in order to better match different application needs.

III. Specializing XSiter for MPEG-7 support and multimedia retrieval

Thanks to the above discussed features, XSiter is already able to provide efficient querying support in very large XML (MPEG-7) digital libraries. Specifically, the high system efficiency with large documents is a particularly crucial requirement for multimedia DLs data querying, due to the huge size of the associated metadata details, especially the ones for videos. However, in order to improve efficacy and efficiency for specific MPEG-7 and multimedia metadata applications, we are working on enhancing our system as follows. The value management will be specialized for managing typical media descriptors. Indeed, as also noticed in [6], large portions of the information contained in MPEG-7 media descriptions typically consist of nontextual data and a system managing them needs to keep the basic contents of media descriptions in typed representation and not just as text. To this end, we will include abstractions needed by the application layer in order to manage complex types such as XML Schema data types, and MPEG-7 specific extensions to them, including arrays and matrices. Specialized index structures that are suitable to manage these additional kinds of data will be included in the system architecture. Notice that additional kinds of indexes, such as the multidimensional ones presented in [1] for numerical data, will also allow XSiter to easily go beyond exact match for values, which is definitely not enough in a multimedia metadata environment, and support *similarity search*. Additional document summaries will also be developed in order to better filter out uninteresting documents or parts of them. Further, some extensions will be included in the GUI subsystem, in particular the user will be helped to specify MPEG-7 queries and audiovisual tools could also be included to present results.

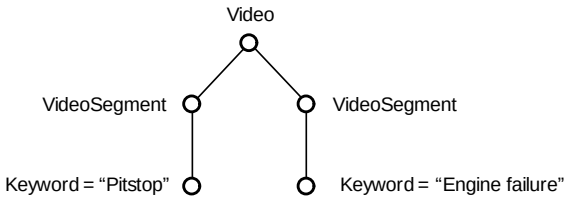
In order to better understand why these changes are useful let us make an example. Suppose that we want to manage an image collection. A typical content media descriptor for images is the color histogram that defines an uniform quantization of the color space. In order to better support an external application working on this data with XSiter, the abstraction of the color histograms should be pushed down to the system, in this way the application could directly interact with the system using domain objects instead of pure values. Since color histograms are multidimensional data, XSiter should be enhanced with multidimensional indexes and the similarity measure should be provided by the application itself.

IV. Possible XSiter use-cases in multimedia DLs

In this section we will briefly describe how some of the features of our system could be successfully exploited in order to accomplish some possible tasks related to multimedia and video DLs querying and management. The considered reference setting will be a Video DL containing Formula 1 video clips and their MPEG-7 metadata.

- The user information need could be quite specific: “retrieve all the videos created by XY”, or “retrieve all the segments involving pitstops”. These are very simple queries which can be straightforwardly managed by XSiter. The key is to efficiently retrieve the occurrences of simple path structures with associated values, such as `//DescriptionMetadata//Creator[.//Name="XY"]` and `//VideoSegment//TextAnnotation[.//Keyword="pitstop"]` and perform simple elaborations on the returned solutions. The expressed relationship between elements are ancestor-descendant ones, which are easily managed by means of the path search algorithms and the underlying signature representation of the documents. As to value support, exact search is in this case required and efficiently supported by means of XSiter value-specialized algorithms, which from our tests are able to perform more than 10 times faster than standard ones for typical cases, when indexed information are available for the most commonly queried elements; in the specific case, inverted indexes are the best option.

- More sophisticated user needs may require XSiter support to more complex query structures, involving ordered and unordered twig patterns. For instance: “Retrieve all videos containing a segment involving a pitstop *followed by* one involving an engine failure”, which can be expressed and evaluated as an *ordered twig*:



  Further, an user could also be interested in a similar but conceptually different query: “Retrieve all videos containing segments involving an engine failure *and* segments involving crashes”. In this case, the structure of the twig pattern would be very similar to the one above, but the order in which the interesting segments appear in the video (and their elements in the MPEG-7 tree) is not important; in other words, the proposed answers should include the video involving the particular segments in *any* order. Another example: “Retrieve all video segments containing all the following cars”, where the cars will be MPEG-7 identified objects, whose relative order in the MPEG-7 annotation is obviously uninteresting for the user. These are computationally more complex problems than ordered matching and can be succesfully solved by exploiting XSiter’s unordered matching algorithms.
- As most multimedia content can not be easily described by linguistic terms and keywords only, more and more next-generation context-based DLs annotations are based on *ontologies*, which contain domain information on the subject. For instance, in [5] a DL based on a pictorially-enriched ontology is presented; such an ontology contains information about typical scenes available in the repository and also provides visual clues of them. By browsing the ontology the user can easily retrieve all video clips related to a particular class, such as “scenes showing track bends”. This can be performed by using simple path (or twig) queries on class annotations (e.g. scenes marked as “class 5”), handled similarly to the ones discussed in the previous points. However, how the videos of the DL can be annotated w.r.t. the ontology classes? This can be achieved by means of similarity queries on the scene feature summaries, involving twig queries with both structural and non-exact value constraints (e.g. “scenes having motion vector similar to X”); in this case the exploitation of multidimensional metric indexes, together with XSiter value-specialized algorithms, appears fundamental for achieving a good efficiency.
- Similarity techniques could also be applied for solving *example-based queries*, very frequent in multimedia DLs. Such queries are also useful for ontology annotated videos: For instance, the ontology could include a class for “track bends”, however the user could be interested in all videos involving a particular bend, the one shown in a user-indicated scene.
- Always in the settings of ontology annotated DLs, by combining the power of value similarity queries together with structural aspects, the ontology could be enriched and structured on different semantic levels, ranging from classes derived from low-level feature similarities to conceptually higher level concepts. For example, based on the information derived from MPEG-7 low-level data like motion descriptors, color istograms, and so on, a first-level class on subshots about “overtakes” could be defined and populated with examples. Then, a second level class, such as “car duels” could be defined by means of structural queries involving first level information (“a scene containing at least two overtake subshots”); then, a further higher level could include a class for “exciting races” videos, defined as the ones involving a high number of duels, and so on. All the associated annotations could be efficiently produced on new video materials w.r.t. the defined ontology by means of a series of structural twig queries as supported by XSiter, and could allow very effective querying possibilities on the DL.

V. Conclusions

Thanks to its extensible architecture and its efficient matching algorithms, our XSiter system is able to provide efficient querying support in very large XML repositories, as we have also experimentally proven through a number of tests, such as the ones presented in [8]. The usefulness of the current system features in a multimedia DL and MPEG-7 environment is already quite relevant; nonetheless, the improvements and specialization we plan to perform on the system should help to quickly develop an even more MPEG-7 aware system without the need to concern about typical XML search problems and not impacting on the main query engine architecture.

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[2] P.F. Dietz. Maintaining Order in a Linked List. In *Proceedings of 14th Anual ACM Symposium on Theory of Computing (STOC)*, pages 122–127, 1982.

[3] T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann. Anatomy of a native xml base management system. *The VLDB Journal*, 11(4):292–314, 2002.

[4] J. M. Martinez. MPEG-7 standard overview, ISO/IEC JTC1/SC29/WG11 N6828. http://www.chiariglione.org/mpeg/standards/mpeg-7.

[5] C. Torniai, A. Del Bimbo, R. Cucchiara, and M. Bertini. Video Annotation with Pictorially Enriched Ontologies. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, 2005.

[6] U. Westermann and W. Klas. An analysis of xml database solutions for the management of mpeg-7 media descriptions. *ACM Comput. Surv.*, 35(4):331–373, 2003.

[7] P. Zezula, G. Amato, F. Debole, and F. Rabitti. Tree Signatures for XML Querying and Navigation. In *Proceedings of the XML Database Symposium (XSym2003)*, pages 149–163, 2003.

[8] P. Zezula, F. Mandreoli, and R. Martoglia. Tree Signatures and Unordered XML Pattern Matching. In *Proceedings of 30th International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM’04)*, 2004.