Annals of the University of Craiova, Mathematics and Computer Science Series
Volume 38(3), 2011, Pages 56–65
ISSN: 1223-6934, Online 2246-9958

Semantic Cache - Optimizing the Semistructured Data Queries

Mihai Stancu

Abstract. The appearance of the XML has raised many problems regarding the efficient obtaining of information from different XML remote sources through Internet. A semantic cache technology can improve the efficiency of the XML data querying process in Web environment. The semantic cache exploits the idea of reusing the query results in order to respond to the new queries based on some inclusion and rewriting techniques. This paper presents the design space of the semantic cache systems and proposes a new improved formula for partial cache replacement strategies.

2010 Mathematics Subject Classification. Primary 60J05; Secondary 60J20.
Key words and phrases. databases, semistructured data, semantic cache.

1. Introduction

The Internet applications and their new methods of data management has changed the role of database theory and its report to the practical issues. This applications have a new data model, the semistructured data model, with syntax offered by XML language. For the XML applications to achieve their maximum potential, we need some adequate tools for processing the data in this format. The data in XML format can be the target of some databases specific operations like: data extraction, data integration, data storage etc. Thus, it has appeared the necessity of defining new query languages for XML data that can offer a maximum support for this kind of operations. This paper is organized as follows. Section 2 presents the basic concepts of XML language and XQuery language. Section 3 describes the semantic cache system used by the XML query engines. Section 4 provides different kinds of cache replacement strategies and proposes a new improved formula for partial cache replacement strategies.

2. Basic Concepts

2.1. XML Language. XML is a markup language for documents with structured informational content developed by W3C (World Wide Web Consortium). The definition of XML specifications can be found on W3C site [18]. The structured information is composed from content and also from some indications regarding the role of that content. All the documents can be organized in some sort of structure. A markup language is a mechanism for identifying a structure within a document. Thus, XML defines a standard for adding the markup tags in a document.

In figure 1 we can see an XML document that describes various existing books within an electronic library. A book is represented by the element <*book*> wich appear as the child of the document's root. First book has an attribute *'year'* with value

Received May 04, 2011. Revision received August 20, 2011.

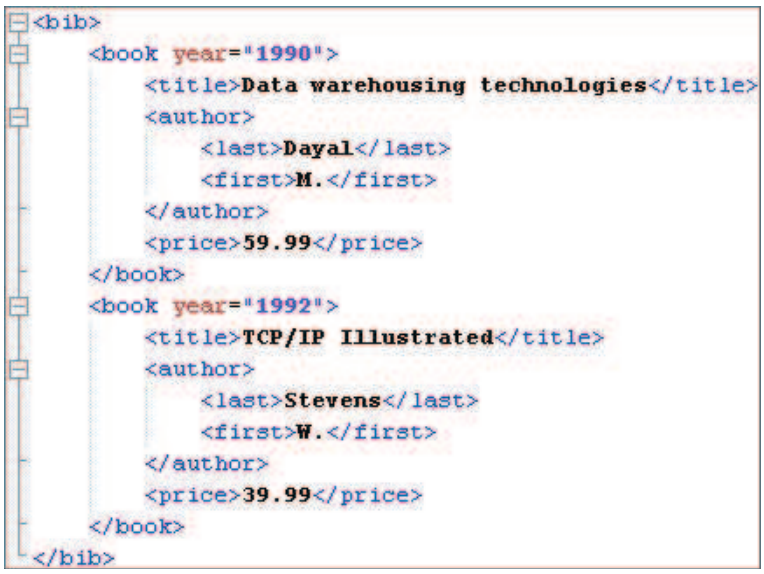

FIGURE 1. bib.xml

'1990'. Each book has a title described by his sub-element <*title*>. The book element has also some others sub-elements representing the author and the price associated to that book. Each of the elements *author* have sub-elements that describe the first name and the last name of the author.

2.2. XQuery. The W3C consortium has proposed a new query language named XQuery, as the standard language for querying XML documents [16]. This language is quite similar with Quilt[1] language and cover functionalities offered by many previous languages like: XPath[2], XQL[3], XML-QL[4], SQL[5]. Thus, XQuery was designed to offer a sufficient number of functionalities to ensure the flexibility needed to be integrated in a large spectrum of XML data sources, including documents but also databases.

The BNF[6] rules that define XQuery language can be observed in figure 2. In these rules, xp denotes an XPath expression, $\{ei\}(i = \overline{1..k})$ denote a sequence of bracket expressions, v is a variable, and c is a condition. By $[v]$ we denote a sequence of variables, and by $[e]$, the corresponding bounded expressions. Within a condition some operators can appear: the existential qualifier SOME, the conjunctive link AND and semi-arithmetic compare operators $(=, \leq, \geq)$. To avoid the recursive calculations, we can consider only the XPath expressions for downward navigation without the disjunction or the negation. Thus, the set of XPath expressions can be denoted by $XP\{//, /, *, []\}$.

[1]J. Robie, D. Chamberlin, D. Florescu, *Quilt: an XML Query Language*, http://almaden.ibm.com/cs/people/chamberlin/quilt_euro.html, 2000.

[2]W3C, *XML Path Language (XPath) 2.0 (Second Edition)*, http://www.w3.org/TR/xpath20/, 2010.

[3]H. Ishikawa, K. Kubota, Y. Kanemasa, *XQL: A Query Language for XML Data*, http://www.w3.org/TandS/QL/QL98/pp/flab.txt, 1998.

[4]W3C, *XML-QL: A Query Language for XML* , http://www.w3.org/TR/NOTE-xml-ql/, 1998.

[5]ISO, *SQL:2008*, ISO/IEC 9075-1, 2008.

[6]Wiki, *Backus-Naur Form*, http://en.wikipedia.org/wiki/Backus-Naur_Form

```
e             ::= v | (v)?xp | FLWR | re
FLWR          ::= FOR WHERE-RETURN
FOR           ::= FOR [v] IN [(v)?xp]
WHERE-RETURN  ::= (WHERE c*)? RETURN e
re            ::= (e1) (..)(ek) | <tag> e </tag>
```

FIGURE 2. BNF rules for XQuery language

3. Cache Systems

3.1. Types of cache. There are three possible cache methods, sometimes named data shipping architectures [7], [10]:

- Page cache
- Tuple cache
- Semantic cache

In a **Page cache** system, the queries are processed on client side until it is decided that a certain page must be requested. Then, a search is performed through the page index on the local cache system. If the page is found, it is returned from the local cache system, otherwise a new request is made to the server in order to get that specific page. The server searches the requested page in the server cache, accesses the disk if necessary, and returns the page to the client.

The **Tuple cache** system is similar to the Page cache with one exception regarding the stored elements, that are tuples instead of pages. This method has the advantage of having more flexibility in the process of cache management. The main disadvantage of this alternative is the fact that sending tuples between server and client is more expensive relative to the performance. Thus, the tuples returned from server are grouped together and send back to the client as tuples collections in order to increase the process efficiency.

Some important aspects about these methods are [4]:

- The tuples are fixed length data elements containing tags and their corresponding data.
- The pages are fixed length physical units.
- There is a fixed amount of tuples in a page; this way a page will have a fixed size.
- Semantic cache uses groups of tuples dynamically defined, so there is no a fixed length used.

The main difference between these three methods lies in the management of the space overload and in the flexibility of grouping technique for the tuples. The tuple cache method resolves the overload problem relative to the tuple numbers that are going to be cached, whereas page cache and semantic cache methods combine the information about some groups of tuples. The semantic cache method offers a complete flexibility regarding the tuples grouping operation that can be adjusted to the specifics of a certain query. With the page cache method, this grouping is static, and independent of the query access pattern.

3.2. Semantic Cache. Semantic Cache manages the client cache by using semantic regions of cache. These regions are somehow similar with pages used in page cache method and correspond to a tuples aggregation. Unlike the pages, the semantic regions have their format and size dynamic.

When a new query appears on the client, this query is broken in two parts as follows:

- **Probe query** - is the part of the query whose result can be served directly from cache.
- **Remainder query** - is the part of the original query whose results are not contained in the local cache and must be received from the server.

The results obtained from these two queries are then combined in pages and then served to the client. Establishing the probe query is done by using the query containment theory [3].

Let's assume that we have a query given by the relation ρ with constraints given by formula Q. With V we will describe the set of tuples from ρ already present in the client's cache. Thus, the probe query denoted by $P(Q, V)$ is defined by the constraint formula $Q \wedge V$ on the relation ρ. The remainder query $R(Q, V)$ is defined by the formula $Q \wedge \neg V$ on the same relation ρ. For a concrete example, let us consider a query that finds all the CDs with a cost bigger than 10 and a classical music style. The query is given by the relation $cd(title, cost, style)$ and the constraints are given by the formula:

$$Q_1 = (cost > 10 \wedge style = classic) \tag{1}$$

Let us consider that the cache already contain all the CDs with costs bigger than 15 . This can be described by the constraint formula:

$$V_1 = (cost > 15) \tag{2}$$

Thus, the probe query is denoted by:

$$P(Q, V) = (style = classic) \tag{3}$$

and the remainder query by:

$$R(Q, V) = (cost > 10 \wedge cost \leq 15 \wedge style = classic) \tag{4}$$

3.3. Query Engine Architecture. A query engine for XML documents contains some modules necessary for processing each input query. Thus, we can say that a query execution has a lifecycle inside a query engine. The stages of this lifecycle are the following [4]:

- Query parsing
- Query decomposition
- Query matching
- Query rewriting
- Query combining

In the query parsing phase, the initial queries are broken in FLWR expressions (For-Let-Where-Return). Additionally, a XQuery normalization technique can be used in order to obtain a canonical form of the FLWR expressions.

The query decomposition defines two tree structures for capturing the data binding hierarchies for the query matching phase and the output view structure for the query combining process. These structures are known as the variable dependency tree (VarTree) and the tagging template tree (TagTree) [4].

The query matching phase uses a tree homomorphism to compare the VarTree obtained in the previous phase with the existing VarTrees from the cache.

In the query rewriting phase the given query is broken in two parts: the probe query - the part of the query whose result will be served directly from cache and the remainder query - the part of the query whose results must be received from the sever.

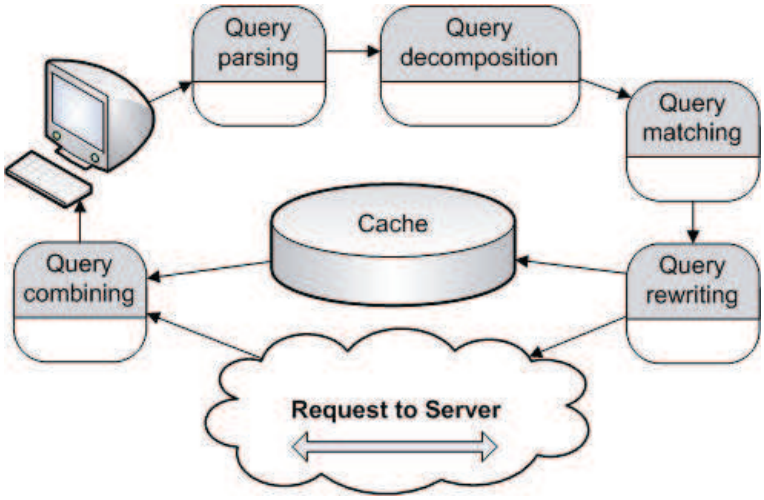

FIGURE 3. Query's lifecycle

In the query combining phase the final result is constructed from the results of these two queries based on the each correspondent TagTree.

4. Cache replacement strategies

The cache is a fundamental concept in modern computational processes. It is successfully used on a large scale within databases, storage systems, Web servers, processors, file systems, physical drives, operating systems etc. A cache system is supposed to be much faster than the secondary storage devices, but, on other hand, it requires a higher maintenance cost. Thus, the storage space required by the cache system is considered to be a limited resource and, therefore, is designed to use only a part from the secondary storage space. After a certain number of requests, the space available to the cache system will be totally used. Considering a full cache, before a new page to be added to the cache, one or more old pages must be purged from cache. These pages are selected using a cache replacement strategy.

4.1. Traditional cache replacement strategies. Lately, many cache replacement strategies have been studied. A common strategy, still very used due to its simplicity, is the replacement strategy of the oldest region, named also LRU (*Least Recently Used*). This strategy is very useful in certain cases when the requested page is very likely to be accessed in the near future [6]. A major disadvantage of this technique is the fact that it can not exploit the regularity of the page accesses; i.e. a sequential access or a cyclic access. This fact brings a degradation of performance in certain cases. Another very often used strategy is to extract the least accessed page, strategy named also LFU (*Least Frequently Used*) [15]. This strategy considers very likely for much accessed pages to be requested in the near future. A possible disadvantage of this strategy may appear in some cases when a page accumulates a big number of accesses and, after this, it will not be accessed for a long period of time. In this scenario, this page is difficult to be extracted from cache.

4.2. Replacement strategies for the semantic cache. A big difference between the semantic cache systems and the traditional ones, based on tuples or pages, is the fact that data are logically organized in queries instead of tuples or pages. Thus, within a semantic cache system, the data granularity necessary to the replacement strategy is represented by the query and the corresponding attached result. The semantic cache manager stores a collection of query regions, each one of these containing the query descriptor and the result as an XML document. The query descriptor is used to establish the inclusion relation between a certain query and a new addressed query. Moreover, the cache replacement strategy can use some statistical information regarding the user accesses attached to the descriptors by computing a utility value for each region.

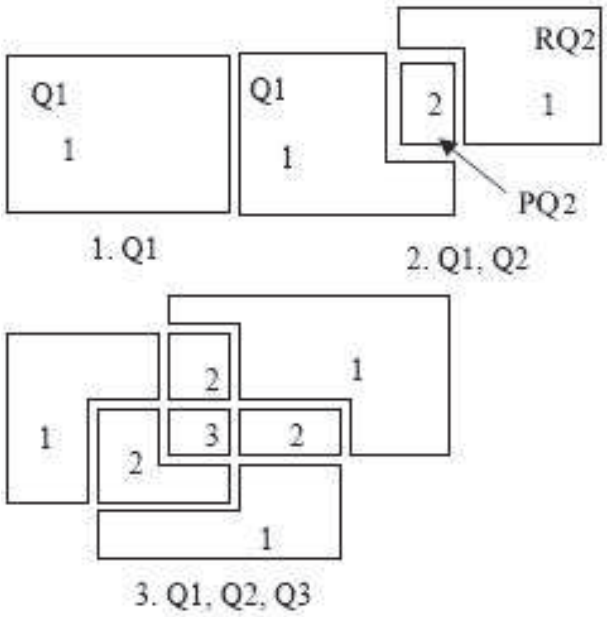

FIGURE 4. Total cache replacement - scheme 1

4.3. Total cache replacement strategies. A new query is conceptually composed of a probe query and a remainder query. The query region that contains the probe query will be split in two regions, one that contains the answer to the probe query and the other one that is the completion of the first relative to the query region. After this splitting, the second region will inherit a uniform utility value from the original region and for the first region a new utility value will be calculated by incrementing the utility value of the original region.

This process is depicted in figure 4. In this figure, the queries are denoted by Q_i, the probe queries by PQ_i, the reminder queries by RQ_i and the utility values are depicted as numbers inside each region. Thus, when the cache is full, we can eliminate a region based on the utility value and, this way, we'll be able to gain more space for a new region. This strategy has the disadvantage of a big number of decompositions and, also, presents a higher rate of fragmentation of the cache regions, a fact that slows down the answering process.

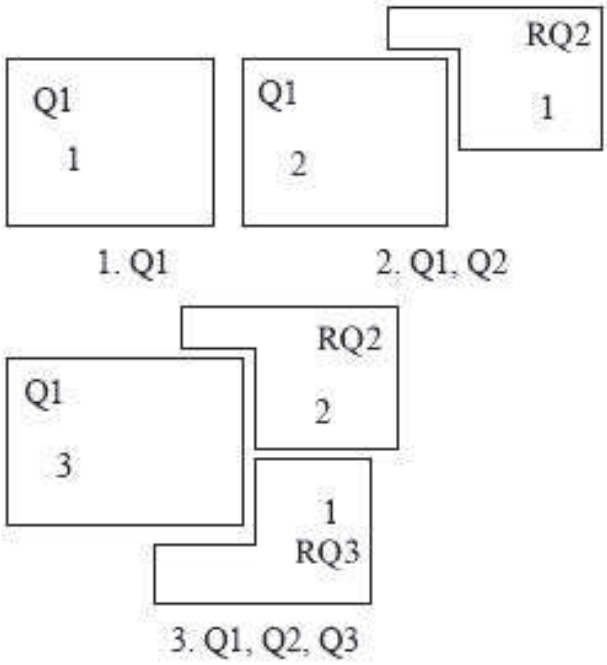

FIGURE 5. Total cache replacement - scheme 2

An alternative strategy is to maintain the query regions without splitting them in sub-regions. Thus, eventually, just one region will be added when a remainder query exist corresponding to some data that are not present in cache. This strategy is presented in figure 5.

This alternative also has some disadvantages. The first one is the fact that the uniform utility value of the regions does not model very well the contribution of different constituent fragments. Another disadvantage is the fact that when a big region is removed from the cache, many possible data will be lost.

4.4. Partial cache replacement strategies. During the past decade, a partial cache replacement strategy was studied in ACE-XQ project [4], [5]. This strategy uses particular utility values to refine the representation of different constituent fragments. Thus, each query descriptor will have attached a table with XPath values for each element returned by the query, along with each particular utility value. Thus, when a query from cache contains the probe query, the particular utility value for all the elements corresponding to the probe query will be updated. When the cache is full, instead of removing a complete region, only the elements with the smallest utility value will be eliminated. The process of removing these elements will be made using a filter query. After the removing operation, the query descriptor will be updated correspondingly. This process is depicted in figure 6.

To compute the utility value we can use the following formula:

$$(\Delta_t * freq * hits) / t_0 \tag{5}$$

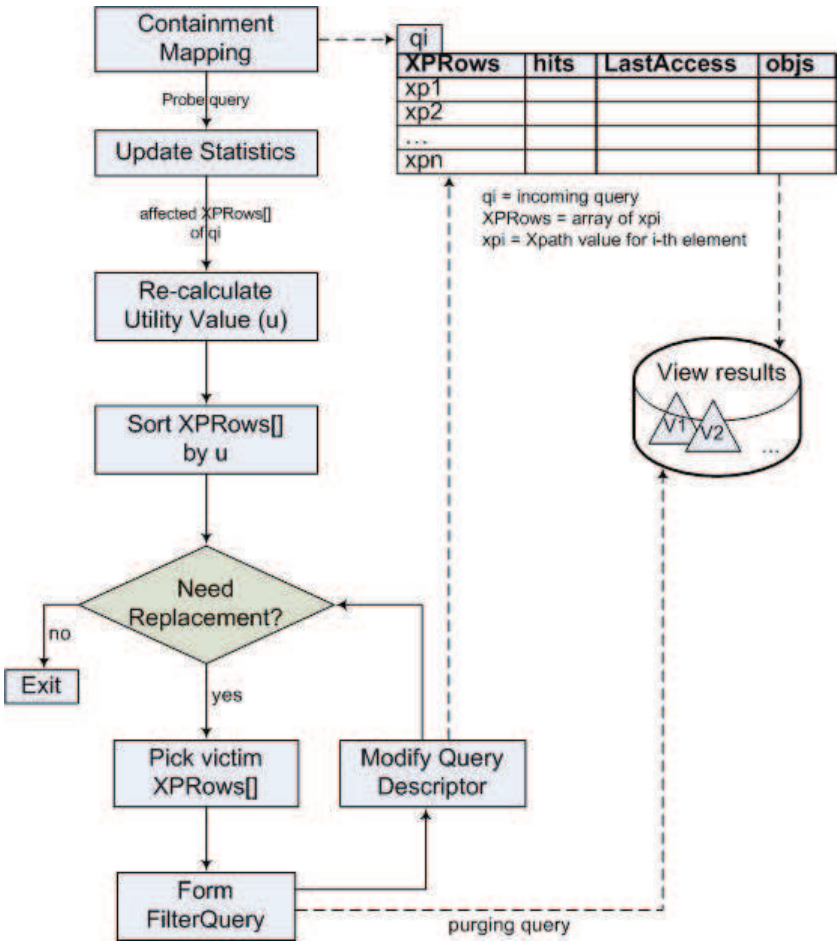

FIGURE 6. Partial cache replacement mechanism

where Δ_t is the time period for the element being present in the cache system, $freq$ is the frequency of his utilizations, $hits$ is the utilizations number, and t_0 is the initial loading time of the cache system.

4.5. Improved formula for partial cache replacement. Sometimes, inside the XML documents, the applications use elements that have attributes of type IDREF to refer some other elements. These references are useful to avoid the data redundancy. Regarding the semantic cache system, the existence of this kind of attributes can be exploited in the sense of modifying the utility value of some elements. When an element is referred by some other elements, its importance can be considered bigger than in a normal scenario. Thus, whereas its standard utility value is small, we will prefer not to remove it from cache with an eventual replacement operation. In this case, we will compute the utility value using the following formula that reflects the importance of its references:

$$((\Delta_t * freq * hits)/t_0) * (k * N_{ref}/N) \tag{6}$$

where N_{ref} is the number of its references, N is the total number of elements, and k is a relevance factor of the references.

Using this strategy we can avoid unnecessary growing of the query regions, but, also, we will keep a much more fine data granularity. Thus, the partial cache replacement strategy succeeds to maintain only the data that are the most likely to be needed in the future queries.

5. Related work

The page caching and tuple caching have been studied primarily in [8]. Alternative approaches for page caching, called Dual Buffering and Hybrid Caching respectively, have been proposed in [12] and [14] in order to balance the tradeoff between overhead and sensitivity to clustering. In [11] was introduced a collection of possibly overlapping constraint formulas, derived from queries, to describe client cache contents. The concept of semantic regions have been primarily introduced in [7]. This semantic regions allows, in particular, the use of sophisticated value functions incorporating semantic notions of locality. In [4] and [5] it is proposed ACE-XQ, a new query system based on XQuery, that aims to minimize the fetching cost of XQuery results by answering new queries using semantic cache. A new fine granularity replacement strategy was also poposed here and it was deployed in ACE-XQ XML query caching system.

6. Conclusions

Due to the growing number of applications that access XML data from different sources, it has become more necessary to efficiently evaluate the XML queries. A big step for obtaining this optimization is the exploit of the cache techniques in order to reduce the response latency caused by sending the data through the Internet.

Unlike the traditional cache systems based on tuples or pages, the semantic cache exploits the idea of reusing the queries and their results by using the inclusion relation. So far, the studies of this relation have been directed more to the XPath expressions [13]. Studying the models to represent the inclusion for a language like XQuery can considerably improve the semantic cache systems. The semantic cache has its data logically organized in queries, instead to use physical tuple identifiers or page numbers. The access to data and the data management is done at the query descriptor level.

The granularity of the data being managed in a semantic cache system is represented, in most cases, by the XML document that is the query result. This thing raises many problems in the cache data management, i.e. cache replacement operations. To free a query from cache means to free the entire associated document, and, in some cases, this file can have a larger size. Recently, some studies have proposed to define the granularity to the XML fragment level, which lead to the partial cache replacement techniques. These techniques give a higher performance to the cache system, allowing an optimal utilization of the storage space.

References

[1] D. Chamberlin, J. Robie and D. Florescu, Quilt: An XML Query Language for Heterogeneous Data Sources, *WebDB* (2000), 53–62.

[2] A.B. Chaudhri, A. Rashid and R. Zicari, *XML Data Management: Native XML and XML-Enabled Database Systems*, Addison-Wesley, 2003.

[3] C. Chekuri and A. Rajaraman, Conjunctive Query Containment Revisited, *ICDT* (1997), 56–70.
[4] L. Chen and E.A. Rundensteiner, ACE-XQ: A CachE-aware XQuery Answering System, *Proceedings of the 5th International Workshop on the Web and Databases (WebDB)* (2002), 31–36.
[5] L. Chen, S. Wang and E.A. Rundensteiner, A Fine-Grained Replacement Strategy for XML Query Cache, *4th Intl. Workshop on Web Information and Data Management (WIDM'02)* (2002), 76–83.
[6] E.G. Coffman and P.J. Denning, *Operating Systems Theory*, Prentice-Hall International Editions, 1973.
[7] S. Dar, M.J. Franklin, B.T. Jonsson, D. Srivastava and M. Tan, Semantic Data Caching and Replacement, *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96)*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1996.
[8] D. DeWitt, P. Futtersack, D. Maier and E. Velez, A study of three alternative workstation-server architectures for object-oriented database systems, *Proc. VLDB Conf.* (1990).
[9] H. Garcia-Molina, J. Ullman and J. Widom, *Database Systems: The Complete Book*, 2nd Ed, Prentice Hall, 2009.
[10] L. M. Haas, D. Kossmann and I. Ursu, Loading a Cache with Query Results, *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1999.
[11] A. Keller and J. Basu, A predicate-based caching scheme for client-server database architectures, *VLDB J* **5** (1996), no. l.
[12] A. Kemper and D. Kossmann, Dual-buffering strategies in object bases, *Proc. VLDB Conf.* (1994).
[13] G. Miklau and D. Suciu, Containment and Equivalence for an XPath Fragment, *Symposium on Principles of Database Systems (PODS)*, Madison, Wisconsin, 2002, 65–76.
[14] J. O'Toole and L. Shrira, Hybrid caching for large scale object systems, *Proc. 6th Wkshp on Pers. Object Sys.* (1994).
[15] J.T. Robinson and M.V. Devarakonda, Data Cache Management Using Frequency-Based Replacement, *SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (1990), 134–142.
[16] W3C, *XQuery1.0: An XML Query Language*. `http://www.w3.org/TR/xquery/`, 2003.
[17] W3C, *XML Query Data Model*. `http://www.w3.org/TR/query-datamodel`, 2000.
[18] W3C, *XML*. `http://www.w3.org/XML`, 1998.

(Mihai Stancu) Department of Computer Science, University of Craiova, 13 A.I. Cuza Street, Craiova, 200585, Romania
E-mail address: `mihai.stancu@yahoo.com`