SMOQE: A System for Providing Secure Access to XML

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Early version, also known as pre-print

Published In:
Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
SMOQE: A System for Providing Secure Access to XML

Wenfei Fan† Floris Geerts† Xibei Jia Anastasios Kementsietsidis
University of Edinburgh
{wenfei@inf, fgeerts@inf, x.jia@sms, akements@inf}.ed.ac.uk

ABSTRACT
XML views have been widely used to enforce access control, support data integration, and speed up query answering. In many applications, e.g., XML security enforcement, it is prohibitively expensive to materialize and maintain a large number of views. Therefore, views are necessarily virtual. An immediate question then is how to answer queries on XML virtual views. A common approach is to rewrite a query on the view to an equivalent one on the underlying document, and evaluate the rewritten query. This is the approach used in the Secure MOdular Query Engine (SMOQE). The demo presents SMOQE, the first system to provide efficient support for answering queries over virtual and possibly recursively defined XML views. We demonstrate a set of novel techniques for the specification of views, the rewriting, evaluation and optimization of XML queries. Moreover, we provide insights into the internals of the engine by a set of visual tools.

1. INTRODUCTION
Views have been widely used in databases to enforce access control, support data integration, and speed up query answering, among other things. For all the reasons that views are essential to traditional databases, XML views are also important for XML data. In many applications, e.g., in XML security enforcement, views are necessarily virtual: a large number of user groups may want to query the same XML document, each with a different access-control policy. To enforce these policies, we may provide each user group with an XML view [3] consisting of only the information that the users are allowed to access, such that users may query the underlying data only through their views. Here the views should be kept virtual since it is prohibitively expensive to materialize and maintain a large number of views, one for each user group.

An immediate question in connection with XML views is how to answer queries posed by users on a virtual view? However desirable, for XML views to be useful in practice this question has to be answered. A common approach (aka. view unfolding) is to rewrite a user query on views to an equivalent one on the underlying document, and evaluate the rewritten query without materializing the view. Nevertheless, the query rewriting is nontrivial. For example, XPath, the core of XQuery and XSLT, is not closed under rewriting, i.e., for an XPath query on a recursively defined view there may not exist equivalent XPath query on the underlying document [4]. This motivates the use of a richer language in the rewriting context and Regular XPath is the most promising candidate for three main reasons. First, Regular XPath is only a mild extension of XPath which supports general Kleene closure (\(\ast\)) instead of the limited recursion ‘//’ (descendant-or-self axis). Therefore, user queries already written in XPath can be used as-is and need not be re-defined, a necessity if a richer language like XQuery or XSLT was used. Second, and more importantly, Regular XPath is closed under rewriting for XML views, recursively defined or not [4]. Since Regular XPath subsumes XPath, any XPath query posed on any XML view can be rewritten to an equivalent Regular XPath query on the underlying data. Third, there is an increasing interest in using Regular XPath as a stand-alone query language, outside the rewriting context.

Given the above, we have developed the Secure MOdular Query Engine (SMOQE), for facilitating the specification of XML views and answering of XML queries on virtual views. The main features of SMOQE are the following.

- SMOQE supports XML views defined by annotating an XML schema with Regular XPath [9] queries, along the same lines as DAD (IBM DB2 XML Extender [6]) and AXSD (Microsoft SQL Server [10] and Oracle [11]). SMOQE supports recursively defined schema (and thus views). It also provides a visual tool, referred to as isMOQE, to help user annotate a schema and define an XML view.
- SMOQE is able to rewrite any Regular XPath query \(Q\) posed by users on a virtual view \(V\) to an equivalent Regular XPath query \(Q'\) on the underlying document \(T\). That is, \(Q'(T) = Q(V(T))\) for any XML document \(T\), where \(V(T)\) would be the XML view if it were materialized. Here \(Q'\) is also in Regular XPath, and is to be executed on the underlying document \(T\) rather than on the view.
- SMOQE encompasses a query engine for Regular XPath queries, implementing an efficient evaluation algorithm and a novel indexing structure.

Existing XML query systems support neither answering (Regular) XPath queries on virtual XML views, nor efficient evaluation of Regular XPath queries. While one can translate Regular XPath to XQuery, this approach is penalized by the overhead of evaluating and optimizing full-fledged XQuery when dealing with much simpler Regular XPath. To our knowledge, SMOQE is the first system that provides efficient support for answering Regular XPath queries over virtual and possibly recursively defined XML views, as well as sophisticated evaluation techniques particularly for Regular XPath.

† Supported in part by EPSRC GR/S63205/01, GR/T27433/01 and BBSRC BB/D006473/1. Wenfei Fan is also affiliated to Bell Laboratories, Murray Hill, USA.
† Floris Geerts is a postdoctoral researcher of the FWO Vlaanderen and is supported in part by EPSRC GR/S63205/01. He is also affiliated to Hasselt University and Transnational University of Limburg, Belgium.
As an immediate application, SMOQE provides a generic, flexible access-control mechanism for XML data, preventing the disclosure of confidential or sensitive information to unauthorized users.

We have fully implemented SMOQE. Leveraging its visual tool isMOQE, the demonstration is to show, step by step, (a) how XML views can be specified by annotating a DTD, (b) how user queries on XML views are rewritten and answered, and (c) how SMOQE optimizes and evaluates Regular XPath queries. It will also demonstrate the efficiency of the evaluation algorithms and the impact of various optimization techniques implemented in SMOQE.

2. SYSTEM ARCHITECTURE

As shown in Fig. 1, SMOQE consists of four major modules: (a) isMOQE, a visual tool through which a user can define XML views, inspect the query rewriting and evaluation, and browse query results (a small user icon is used to indicate all the system components accessible through isMOQE); (b) a query rewriter (indicated by a box at the left of the figure) for translating user Regular XPath queries posed on XML views to equivalent Regular XPath queries on the underlying document; (c) a query evaluator (indicated by a box in the middle of the figure) for processing Regular XPath queries; and (d) an indexer (indicated by a box at the right of the figure), which is used by the evaluator to build indexes and optimize queries.

XML view definition. SMOQE supports two view definition modes. One mode allows users to define an XML view by leveraging isMOQE to annotate a view schema. The other mode is by means of automated view derivation as proposed in [3]: for each user group, an authorized security administrator annotates the document schema to specify the part of information that users are granted or denied access to, using simple boolean predicates; then SMOQE automatically translates the specification to the definition of a (possibly recursively defined) XML view, along with a view schema that is exposed to the users.

Query support. SMOQE supports Regular XPath in two query evaluation modes: a user may pose a query either (a) directly on the underlying XML document provided that the user is granted access to it, or (b) on an XML view specified for the group which the user is in. In the former case, the evaluator processes the query on the underlying document, capitalizing on the Indexer. In the latter case, the user query is first translated to an equivalent query on the underlying document, and then the rewritten query is answered by the evaluator, without materializing the view.

XML documents. SMOQE supports two modes: a DOM mode and a SAX mode. A DOM mode allows the complete structure of the document to be loaded into memory, while a SAX mode only processes data events (e.g., start element, end element). SMOQE supports two view definition modes.

3. DEMONSTRATION OVERVIEW

The demonstration aims to show the following: (a) how users may define XML views by means of schema annotation, with the aid of isMOQE; (b) how SMOQE answers Regular XPath queries posed on a virtual XML view by using the rewriter, without materialization; (c) how the evaluator of SMOQE processes Regular XPath queries; (d) how the indexing structure of SMOQE helps query optimization and processing; and (e) how isMOQE helps users browse the query result as well as help implementers monitor query processing. These provide a complete picture for how one can leverage SMOQE to enforce XML access control (via view definition and view query answering) and evaluate Regular XPath queries, among other things. Below we present a brief introduction to the techniques that are demonstrated.

Specifying XML views. Like DAD [6] and AXSD [10, 11], SMOQE supports XML views by means of an access control policy which annotates a schema with Regular XPath expressions. For example, Fig. 3(a) shows a schema for a hospital DTD, while Fig. 3(b) shows an access control policy that only exposes the records of patients that took medication for “autism”. Notice that for security reasons, the policy hides the names and test information of these patients. Given such a policy, SMOQE automatically generates the view specification and view DTD shown in Fig. 3(c) and Fig. 3(d), respectively. Conceptually, an XML view defined in this way uses the Regular XPath queries in the specification to extract data from the underlying document, and populate the view using the extracted data, strictly following the schema. Although no actual view materialization occurs, the procedure assures that the view makes sense, i.e., it conforms to the view schema. A unique feature of the SMOQE view language is that it allows the schema to be recursive, and thus supports recursively defined XML views.
We demonstrate how users can leverage SMOQE to define a view. As shown in Fig. 2, SMOQE supports a visual view specification tool that provides the user with an XML schema graph. The user can click on any node (element type) in the graph, and input a Regular XPath query annotating the corresponding elements.

Rewriter. While it is always possible to rewrite a Regular XPath query to a view on an equivalent query Q′ on the underlying document, the size of Q′, if directly represented as Regular XPath expressions, may be exponential in the size of Q [4]. The SMOQE rewriter overcomes the challenge by employing an automaton characterization of Q′, denoted by MFA (mixed finite state automaton) [4], which is linear in the size of Q. An MFA of Q′ is a finite state automaton (NFA, characterizing the data-selection path of Q′) annotated with alternating automata (AFA, capturing the predicates of Q′). For example, Fig. 4(a) depicts the MFA M0 characterizing the Regular XPath query:

\[ Q_0 = \text{hospital/patient}([\text{parent/patient}]) / \text{visit/treatment/test/} \text{and visit/treatment[medication/text()=\textquoteleft\textquoteleft\textquoteright\textquoteleft\textquoteright\textquoteleft\textquoteright\textquoteright\textquoteleft\textquoteright\textquoteright\textquoteleft\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquoteright\textquote...
4. REFERENCES


The demonstration will also show another feature of \textit{iSMOQE}: \textit{iSMOQE} is able to mark nodes in an XML document (in the tree mode) with different colors, indicating whether or not a node is visited during the query evaluation, whether or not it is put in the auxiliary structure \textit{Cans}, and which optimization techniques contribute to its pruning if it is not in the answer of the query. This opens a window to the blackbox of query processing, allowing one to assess the effectiveness of various optimization techniques.

Summing up, we demonstrate the support of \textit{iSMOQE} for different XML view specification methods, its ability to evaluate Regular XPath queries, its capability of answering Regular XPath queries posed on virtual XML views without materialization, the efficiency of the \textit{iSMOQE} evaluator and the effectiveness of the \textit{iSMOQE} indexer. Furthermore, \textit{iSMOQE} visualizes the connection between Regular XPath queries and automata representation, the index structure built on XML data, the huge nodes pruning when the automata are running, and the contributions of different optimization techniques to the pruning.