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 query 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 $(.)^*$ 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.

VLDB '06, September 12-15, 2006, Seoul, Korea.

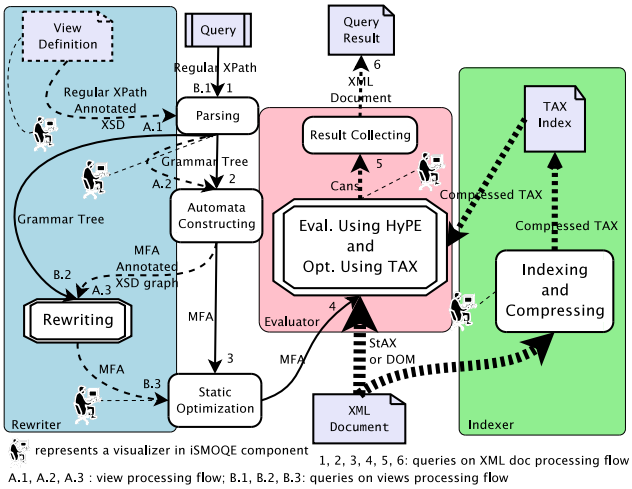


Figure 1: The SMOQE Architecture

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 *i*SMOQE, 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) *i*SMOQE, 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 *i*SMOQE); (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 *i*SMOQE 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 the 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 StAX [7] (Streaming API for XML, a new standard API for XML pull parsing, to be included in Java6) mode. In the DOM mode, the whole document tree will be loaded into memory in order to evaluate a query. One the other hand, in StAX mode the document does not need to be loaded into memory and only one sequential scan of the document from disk is needed for the evaluation. The StAX mode allows to process larger documents efficiently and offers significant advantages over main-memory XPath engines such as Xalan [13] and Saxon [12], which need to randomly access the document during evaluation.

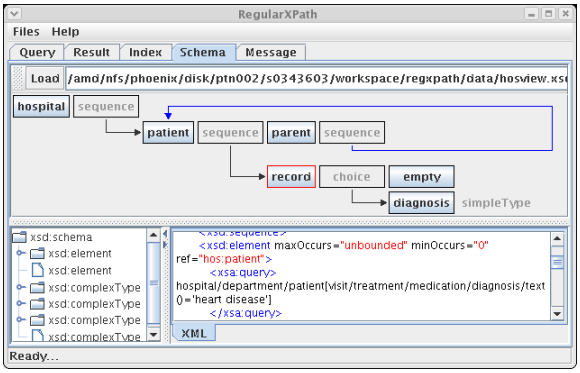


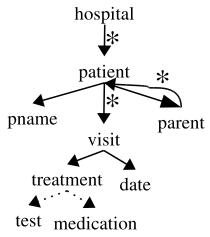
Figure 2: The visual tool in *i*SMOQE for specifying views

Visual aid. *i*SMOQE is the front-end that both provides a user-friendly interface to SMOQE, and it opens a window of the system to let user visually monitor the internals of the engine. It consists of a graphical querying interface, a semi-automatic view definition tool, and query, automaton, index and result visualization tools.

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 *i*SMOQE; (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 *i*SMOQE 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 of SMOQE for supporting these functionalities, as well as a more detailed description of the demonstration *w.r.t.* each of these.

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.



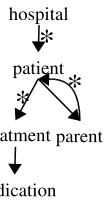
(a) document DTD D

production: *hospital → patient**
ann(hospital,patient) = [visit/treatment/ medication = 'autism']
production: *patient → pname, visit*, parent**
ann(patient, pname) = N
ann(patient, visit) = N
production: *parent → patient*
production: *visit → treament, date*
ann(visit, treatment) = [medication]
production: *treatment → test + medication*
ann(treatment, test) = N

(b) access control policy S_0

production: *hospital → patient**
σ_0*(hospital, patient) = patient[visit/ treatment/medication = 'autism']*
production: *patient → treatment*, parent**
σ_0*(patient, treatment) = visit/treatment [medication]*
σ_0*(patient, parent) = parent*
production: *parent → patient*
σ_0*(parent, patient) = patient*
production: *treatment → medication*
σ_0*(treatment, medication) = medication*

(c) view specification σ_0 derived from S_0



(d) view DTD D_V

Figure 3: Enforcement of access control by security views

We demonstrate how users can leverage *i*SMOQE to define a view. As shown in Fig. 2, *i*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 Q on a view to 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 $\mathcal{M}_0$ characterizing the Regular XPath query:

Q_0 = hospital/patient[(parent/patient)* /visit/treatment/test/
and visit/treatment[medication/text()="headache"]]/pname

In the MFA $\mathcal{M}_0$, the NFA consists of states $(0, 1, 3, 24)$ and represents the selection path *hospital/patient/pname*; it is annotated with an AFA (linked to state 3 via a dotted arrow) capturing the predicate of Q_0 (the part enclosed in [ ]). The notion of MFA is proposed by SMOQE to characterize Regular XPath queries. It is quite different from automata developed for XPath and XML stream processing (*e.g.*, tree automata of [8], XFilter [1], YFilter [2], XPush machine [5]).

The demonstration will show the following, which are visualized by means of *i*SMOQE.

• The MFA characterization of Regular XPath queries. Given a Regular XPath query Q, SMOQE automatically generates a MFA characterizing Q. As an example, Fig. 4(b) displays the MFA $\mathcal{M}_0$ of the query Q_0 given earlier.

• The process of query rewriting. Given an XML view definition V and a Regular XPath query Q posed on V, *i*SMOQE demonstrates how the SMOQE rewriter works by displaying the MFA representation of the rewritten query Q', which is automatically generated by the rewriter, and is equivalent to Q when being executed on the underlying document.

Evaluator. The SMOQE evaluator implements a novel algorithm for processing Regular XPath queries represented by MFA's. The algorithm, referred to as HyPE (Hybrid Pass Evaluation) [4], takes an MFA as input and evaluates it on an XML tree. A unique feature of HyPE is that it needs a single top-down depth-first traversal of the XML tree, during which HyPE both evaluates predicates of the input query (equivalently, AFA of the MFA) and identifies potential answer nodes (by evaluating the NFA of the MFA). The potential answer nodes are collected and stored in an auxiliary structure, referred to as Cans (candidate answers), which is often much smaller than the XML document tree. After the traversal of the document tree, HyPE only needs a single pass of Cans to select the nodes that are in the answer of the input query. This is the reason why SMOQE is capable of efficiently processing Regular XPath queries no matter whether it is in the DOM mode or in the StAX mode.

To our knowledge, previous systems require at least two passes of XML tree traversal to evaluate even XPath queries. For example, to evaluate an XPath query q on an XML document T, Arb [8] requires a bottom-up pass of T to evaluate all the predicates of q, followed by a top-down pass to evaluate the selecting path of q. It uses tree automata, which are more complex than MFA and require a pre-processing step (another scan of T) to parse the document and convert it to a special data format (a binary representation of T). In contrast, SMOQE is able to evaluate Regular XPath queries, more complex than XPath queries. The SMOQE evaluator requires neither pre-processing of the data nor the construction of tree automata. It only needs a single pass of the document during which it often prunes a large number of nodes that do not contribute to the answer of the query.

In the demonstration we show the following.

• The efficiency of the SMOQE evaluator. We show that SMOQE is capable of efficiently evaluating Regular XPath queries, in both the DOM mode or the StAX mode. Furthermore, it outperforms popular XPath engines such as Xalan [13].

• The insight of Algorithm HyPE. Using *i*SMOQE we reveal the details of the evaluation of Regular XPath queries (MFA). For example, Fig. 5 shows the evaluation of the MFA $\mathcal{M}_0$ given earlier on an XML document. It demonstrates how $\mathcal{M}_0$ traverses the document and which nodes are selected and stored in Cans.

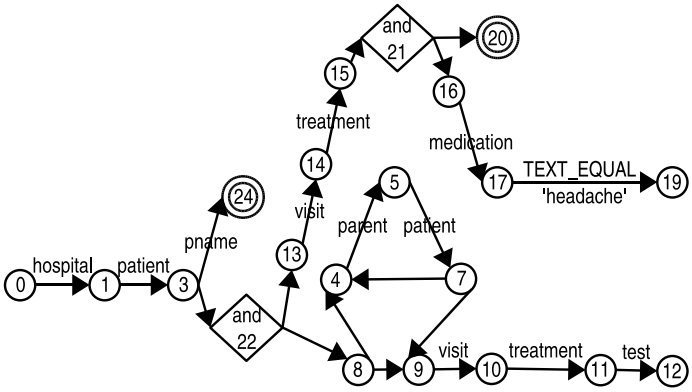
Indexer. SMOQE proposes and implements a new indexing structure, referred to as TAX (*Type-Aware XML index*) [4], to optimize query processing. The novelty of TAX is that it classifies the information of descendants of each node based on their element types. While several labeling and indexing techniques were developed for optimizing the evaluation of XPath queries, they focus mainly on optimizing the evaluation of '//' (descendant-or-self axis) by testing efficiently whether, given two nodes, one is a descendant of the other. As such, they are limited in scope. In contrast, TAX is effective in pruning large document subtrees during the evaluation of XPath queries with or without '//', by keeping track of descendants of certain types that have been and have not been checked at each node. The SMOQE indexer constructs the TAX index, compresses it before it is stored in disk, and uploads it from disk when needed.

The demonstration shows the following.

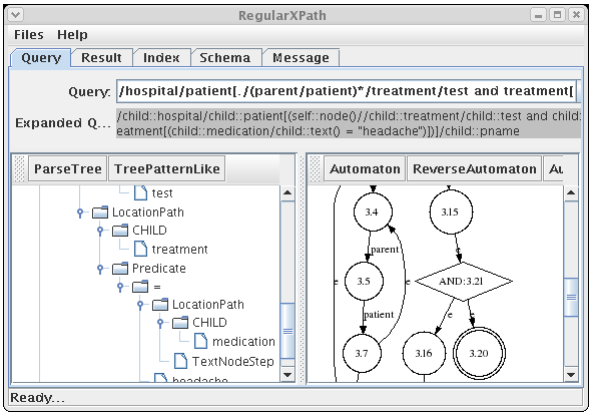
• The effectiveness of TAX. It demonstrates the impact of TAX on the performance of the evaluator by turning on the indexer versus the setting when the indexer is off.

• The insight of TAX. *i*SMOQE is able to show how the SMOQE indexer builds TAX on an XML document. For example, Fig. 6 is an *i*SMOQE display of TAX on an XML tree.

The output visualizer. *i*SMOQE is able to display the output of the query evaluation in two modes: (a) the text mode, which presents



(a) The MFA $\mathcal{M}_0$ for Q_0



(b) The tool in *i*SMOQE for visualizing query and automaton

Figure 4: The MFA $\mathcal{M}_0$ characterizing query Q_0

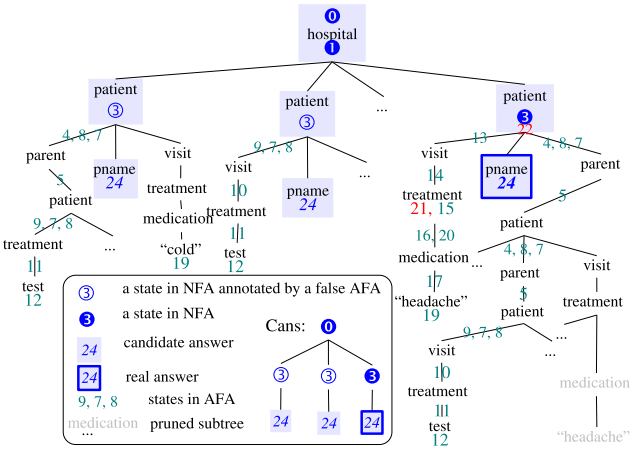


Figure 5: Evaluation of $\mathcal{M}_0$ using HyPE

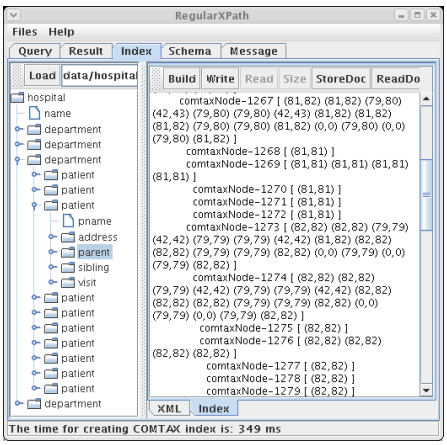


Figure 6: TAX index

the answer of the query as a document in XML syntax; (b) the tree mode, which displays the answer of the query as a tree; it provides an interactive interface such that users may click on a node to browse its subtree.

The demonstration will also show another feature of *i*SMOQE: *i*SMOQE 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 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 SMOQE 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 SMOQE evaluator and the effectiveness of the SMOQE indexer. Furthermore, *i*SMOQE 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.

4. REFERENCES

[1] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB*, 2000.

[2] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *TODS*, 28(4):467–516, 2003.

[3] W. Fan, C. Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD*, 2004.

[4] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Rewriting regular XPath queries on XML views. *http://www.lfcs.inf.ed.ac.uk/research/database/rewriting.pdf*.

[5] A. K. Gupta and D. Suciu. Stream processing of XPath queries with predicates. In *SIGMOD*, 2003.

[6] IBM. DB2 XML Extender. *http://www-3.ibm.com/software/data/db2/extended/xmlext/*.

[7] JSR 173. Streaming API for XML. *http://www.jcp.org/en/jsr/detail?id=173*.

[8] C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *VLDB*, 2003.

[9] M. Marx. XPath with conditional axis relations. In *EDBT*, 2004.

[10] Microsoft. XML support in microsoft SQL server 2005, December 2005. *http://msdn.microsoft.com/library/en-us/dnsql90/html/sql2k5xml.asp/*.

[11] Oracle. Oracle Database 10g Release 2 XML DB Technical Whitepaper. *http://www.oracle.com/technology/tech/xml/xmldb/index.html*.

[12] SAXON. The XSLT and XQuery processor. *http://saxon.sourceforge.net*.

[13] Xalan. *http://xalan.apache.org*.