Efficient Database-Driven Evaluation of Security Clearance for Federated Access Control of Dynamic XML Documents

Erwin Leonardi[1] Sourav S. Bhowmick[1] Mizuho Iwaihara[2]

[1]School of Computer Engineering, Nanyang Technological University, Singapore
[2]Graduate School of Information, Production and System, Waseda University, Japan
assourav@ntu.edu.sg, iwaihara@aoni.waseda.jp

Abstract. Achieving data security over cooperating web services is becoming a reality, but existing XML access control architectures do not consider this federated service computing. In this paper, we consider a federated access control model, in which *Data Provider* and *Policy Enforcers* are separated into different organizations; the *Data Provider* is responsible for evaluating criticality of requested XML documents based on co-occurrence of security objects, and issuing security clearances. The *Policy Enforcers* enforce access control rules reflecting their organization-specific policies. A user's query is sent to the *Data Provider* and she needs to obtain a permission from the *Policy Enforcer* in her organization to read the results of her query. The *Data Provider* evaluates the query and also evaluate *criticality* of the query, where evaluation of sensitiveness is carried out by using *clearance rules*. In this setting, we present a novel approach, called the DIFF approach, to evaluate security clearance by the *Data Provider*. Our technique is build on top of relational framework and utilizes pre-evaluated clearances by taking the differences (or deltas) between query results.

1 Introduction

Increasingly, data and services over the Web are becoming decentralized in nature. For example, the architecture of web services is becoming more decentralized; a number of servers stretching over different locations/organizations are orchestrating together to provide a unified service, sometimes referred to as *cloud computing* [5]. In this setting, access control for protecting sensitive data in XML format should also be cross-organizational, where a user, an access requester, and the *Data Provider* holding sensitive data, belong to different organizations. Figure 1 shows a conceptual depiction of such a federated access control model. The *Data Providers* and *Policy Enforcers* are separated into different organizations; each *Data Provider* is responsible for evaluating criticality of requested XML documents based on co-occurrence of security objects, and issuing security clearances. The *Policy Enforcers* enforce access control rules which reflect their own organization-specific policies. We assume that these organizations have agreed on *global* security policies for information exchange.

Let us illustrate the architecture with an example depicted in Figure 2(a) containing a part of a travel plan produced by a travel agency. We assume that the travel agency respond to request from clients and users using XML documents. These documents may

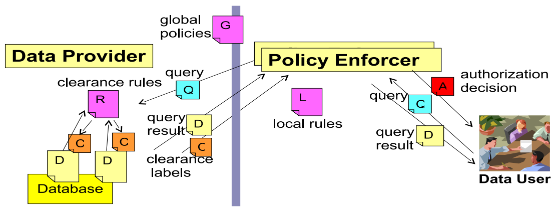
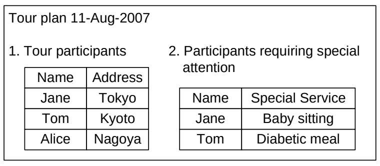


Fig. 1. Overview of federated access control.


Tour plan 11-Aug-2007

1. Tour participants

Name	Address
Jane	Tokyo
Tom	Kyoto
Alice	Nagoya

2. Participants requiring special attention

Name	Special Service
Jane	Baby sitting
Tom	Diabetic meal

(a) A Published Document

objects	label
{Alice, Nagoya}	*L1*
{Jane, Tokyo}	*L2*
{Tom, Kyoto, Diabetic meal}	*L3*

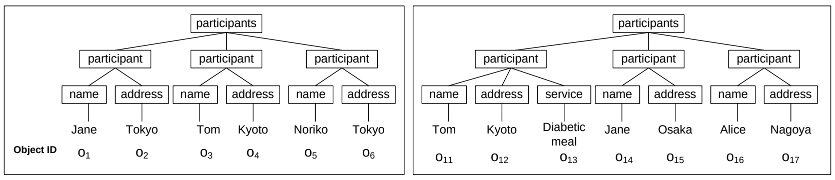
(b) Clearance Rules

Fig. 2. Example.

contain sensitive information. Suppose that an XML document containing relevant results is requested by a user in an airline company providing flights for the tour. The user needs to obtain a permission from the *Policy Enforcer* in his/her organization to read the document. The user query is sent to the *Data Provider* (in this case, the travel agency). The *Data Provider* evaluates the query and also evaluate criticality of the query, where evaluation of sensitiveness is carried out by using *clearance rules* R. A *clearance rule* $r \in R$ is a 2-tuple $[O, L]$, where O is a set of objects existing in XML documents and L is a *clearance label* that defines necessary security clearance the user should have. In this paper, we limit the scope of the objects to be text nodes of an XML tree. Note that these objects can be results of a set of XPath queries. A rule r raises a security caution defined by L iff $O \subseteq B$ where $B = \{b_1, b_2, \ldots, b_n\}$ is a bag of objects in a query result q. Figure 2(b) illustrates a sample of clearance rules represented as a table. If we apply the clearance rules to the document shown in Figure 2(a), we obtain the clearance labels $L1, L2$, and $L3$. For instance, the objects `Alice` and `Nagoya` appear in the document and matches the rule `(Alice, Nagoya,` $L1$`)`. Likewise, `Jane` and `Tokyo` appear in the document and matches the rule `(Jane, Tokyo,` $L2$`)`. The co-occurrence of `Tom`, `Kyoto`, `Diabetic` meal raises the label $L3$. A partial order between labels (such as $L1 < L2 < L3$) can be introduced, where '$A < B$' means that B is superior or more cautious than A. A query may raise a set of clearance labels, but if a priority order between labels is defined, a label that is dominated by another superior label can be ignored.

Finally, the *Policy Enforcer* receives the clearance labels C, and decides whether the user is eligible for the clearance by mapping the labels C to its local roles, and checking whether the user is assigned to one of these roles. Observe that by issuing clearance, the *Data Provider* can export the task of access authorization to the *Policy Enforcer*, thus realizing federated access control.

Motivation and overview. There has been a number of efforts to realize federated access control [1–3, 6, 8]. However, to the best of our knowledge, none of these efforts



(a) Document q_1 (b) Document q_2

Fig. 3. Example of results evaluated by the *Data Provider*.

Document (DocID, DocName)
Path (PathID, PathExp)
PathValue (DocID, LeafOrder, PathID, SiblingOrder, IxSibLevel, LeafValue)
AncestorInfo (DocID, NodeLevel, MinSibOrder, MaxSibOrder, NodeName)

(a) SUCXENT Schema

Rules (RuleNo, ObjID, Object, Label, TotalObjects)
FoundVRules (RuleNo, ObjID, Counter)
FirstFoundVRules (RuleNo, ObjID, Counter)
Del_Obj (Value, DelCounter)
Ins_Obj (Value, InsCounter)

(b) Additional Relations

Rule No	Obj ID	Object	Label	Total Objects
1	1	Alice	L1	2
1	2	Nagoya	L1	2
...	...	...	...	...

(c) The Rules Table (Partial view)

Fig. 4. Relational schemas.

have undertaken a systematic study of the security clearance technique supported by the *Data Provider* in a **dynamic** environment where the underlying XML documents may evolve with time. In this paper, we propose *a database-driven, diff-based strategy to address this issue*. Our proposed technique compliments existing research on federated access control strategies for XML documents.

Since the access control policies are realized through integration of the clearance rules at the *Data Provider* and the local rules at *Policy Enforcers*, at first glance, it may seem that we could take the strategy of pre-evaluating clearance rules at the *Data Provider* as much as possible and cache obtained clearance labels. The advantage of this strategy is that clearance evaluation for repeated queries can be avoided. However, this approach is not a feasible strategy as illustrated by the following example. Consider the documents q_1 and q_2 sent by the *Data Provider* to a *Policy Enforcer* in response to a user's queries at times t_1 and t_2, respectively, where $t_1 < t_2$. We assume that the *Data Provider* represents the query results in XML format and the set of clearance rules in Figure 2(b) must be satisfied by the documents. It is quite possible for q_1 and q_2 to share some data objects due to the following reasons: (a) q_1 and q_2 are results of the *same* query that is issued at times t_1 and t_2. The results may not be identical as the underlying data have evolved during this time period; (b) q_1 and q_2 are results of two different queries. However, some fragments of the underlying data may satisfy both the queries. Consequently, only the second rule in Figure 2(b) is valid for q_1. However, in q_2 this rule does not hold anymore. On the contrary, now the first and third rules are valid for q_2. In other words, updates to the underlying data invalidates caching of the clearance rules of q_1.

In this paper, we take a novel approach for evaluating security clearance by exploiting the overlapping nature of query results. Specifically, we investigate taking differences (deltas) of XML representations of the query results, so that valid clearance labels can be detected and reused. We compute the clearance labels of the first result (q_1) by scanning the entire result. Subsequently, labels of subsequent results are computed efficiently by analyzing the differences between the results. We refer to this strategy as the DIFF approach. Since we store the clearance rules and XML results in a RDBMS, the DIFF approach detects differences in the query results and clearance rules using a series of SQL statements. In the next section, we elaborate on this approach. Note that due to space constraints, the naïve approach of scanning the entire resultset for *every* request (referred to as the SCAN approach) is discussed in [7].

2 The DIFF Approach

Consider a set of query results $Q = \{q_1, q_2, \ldots, q_n\}$ in XML format. We refer to these results as *versions* in the sequel. Assume that the clearance labels for q_1 are cached, but no cache entry exists for the remaining results q_i where $i > 1$. How can the *Data Provider* evaluate clearance labels for the remaining $(n-1)$ versions efficiently? In the DIFF approach, we take advantage of the significant overlaps between q_1 and remaining results by reusing cached clearance labels whenever possible, and re-evaluate the clearance rules that are only affected by the changes (deltas) to the results. Note that often the size of the deltas are typically smaller than the size of q_i.

2.1 Relational Schema

We first present the relational schema that we use for storing results and clearance rules in the database for both SCAN and DIFF approaches. As the results requested by a *Policy Enforcer* are represented in an XML format, we can use any existing techniques for XML storage built on top of a RDBMS [4] to store these results. We use the SUCXENT schema [9] depicted in Figure 4(a) for storing the request results in a RDBMS. SUCXENT is a tree-unaware approach for storing and querying XML documents in relational databases. Particularly, in this paper, only the `LeafValue` attribute of the `PathValue` table is used for security clearance evaluation. The `PathValue` table stores the textual content of the leaf nodes of an XML tree in the `LeafValue` column. Hence, we do not elaborate on the remaining attributes and tables in Figure 4(a).

The clearance rules are stored in the `Rules` table (Figure 4(b)). The `RuleNo` attribute is used as an unique identifier of a rule. The `TotalObjects` attribute maintains the total number of sensitive objects in a rule r whose co-occurrences raise security cautions. The level of security caution is stored in the `Label` attribute. The `ObjID` and `Object` attributes store the identifier and value of the text objects in the query results, respectively. For example, Figure 4(c) depicts how the first rule `(Alice, Nagoya,` $L1$`)` in Figure 2(b) is stored in the `Rules` table. The `FoundVRules` and `FirstFoundVRules` tables are used to keep track of the number of sensitive objects that appeared in the requested query results. The number of occurrences of k-th sensitive object of a rule r is stored in the `Counter` attribute. The remaining tables shall be elaborated in Section 2.3.

2.2 Effects of the Changes

Suppose we have two query results, namely q_1 and q_2, and a set of clearance rules $C = \{c_1, c_2, \ldots, c_n\}$. After evaluating q_1, let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of clearance rules that match with q_1 where $R \subseteq C$. Let O_{q1} and O_{q2} be the bags of objects in q_1 and q_2, respectively.

Let us now discuss the effects of the changes to the query results on the clearance rules. In this paper, we focus on two types of change operations to the query results:

```
Input:
  - Q is the PathValue table
  - R is the Rules table

Output:
  - Z is pairs of query id
        and violated rules

01  initialize Z, C
02  for each qi in Q do
03    if qi is the first
            query result then
04      C = evaluateDoc(qi, R)
05      FR = C.clone()
06    else
07      C = FR.clone()
08      D = findChanges(q1, qi)
09      C = evaluateDelta(D, R)
10    end if
11    V = findViolatedRules(C)
12    Z.add(i, V)
13  end for
14  return Z
```

(a) The DIFF Algorithm

```
01  INSERT INTO FoundVRules
02  SELECT DISTINCT RuleNo, ObjID, 0
03  FROM Rules
```

(b) Initialize the *FoundVRules* Table

```
01  UPDATE FoundVRules
02  SET COUNTER = C.TOTAL
03  FROM FoundVRules F,
04     (SELECT  C.RULENO, C.OBJID,
                COUNT(*) AS TOTAL
05      FROM RULES C, PATHVALUE L
06      WHERE L.DOC_ID = did
07        AND CHARINDEX(
               ' ' + C.OBJECT + ' ',
               ' ' + L.LEAFVALUE +' ') >0
08      GROUP BY C.RULENO, C.OBJID) C
09  WHERE C.RULENO = F.RULENO
10    AND C.OBJID = F.OBJID
```

(c) Evaluate First Query Result

```
01  TRUNCATE TABLE FirstFoundVRules
02  INSERT INTO FirstFoundVRules
03  SELECT * FROM FoundVRules
```

(d) Clone Tables

```
01  SELECT L.LEAFVALUE,
           COUNT(*) AS COUNTER
02  FROM
03    (SELECT L1.LEAFVALUE
04     FROM PATHVALUE L1
05     WHERE L1.DOC_ID = did1
06     EXCEPT ALL
07     SELECT L2.LEAFVALUE
08     FROM PATHVALUE L2
09     WHERE L2.DOC_ID = did2) L
10  GROUP BY L.LEAFVALUE
```

(e) Find the Changes

Fig. 5. The DIFF algorithm and SQL queries.

deletion and *insertion* of text objects. Note that the *update* of a text object can be represented as a sequence of delete and insert operations. An object o_{del} is a **deleted** object iff $o_{del} \in O_{q1}$ and $o_{del} \notin O_{q2}$. Similarly, an object o_{ins} is an **inserted** object iff $o_{ins} \notin O_{q1}$ and $o_{ins} \in O_{q2}$.

Property 1. A deletion of an object o_{del} will cause the removal of clearance rule $r \in R$ iff co-occurrence o_{del} with $o_k \in O_{q1}$ forms the clearance rule $r \in R$, and there does not exist $o_{k'} \in O_{q1}$ such that $value(o_{k'}) = value(o_{del})$ where $value(o)$ is the text value of object o.

Property 2. An insertion of an object o_{ins_1} will cause an addition of clearance rule r into R if o-occurrence of o_{ins_1} with $o_j \in O_{q1}$ forms a clearance rule $r \in C$, or co-occurrence of o_{ins_1} with another inserted object o_{ins_2} forms a clearance rule $r \in C$.

2.3 The DIFF Algorithm

The DIFF algorithm is depicted in Figure 5(a). The input to the algorithm are two relational tables, namely the `PathValue` table (denoted as Q in Figure 5(a)) and the `Rules` table (denoted as R in Figure 5(a)). Note that the requested results are stored in the `PathValue` table. The first step is to initialize the `FoundVRules` table (denoted as C in Figure 5(a)) by invoking an SQL query depicted in Figure 5(b) and a list Z. For each query result, the algorithm will do the followings (Lines 02–13). If the current query result is the first one (q_1), then it evaluates the occurrences of sensitive objects in q_1 (Lines 03–05). The evaluation is done by executing the SQL query depicted in Figure 5(c). The objective of this query is to update the value of `Counter` attribute of the `FoundVRules` tables to the number of occurrences of a sensitive object in a particular rule (Lines 04-08, Figure 5(c)). Next, the algorithm clones the `FoundVRules` table into the `FirstFoundVRules` table. The `FirstFoundVRules` table stores the results generated by evaluation of q_1.

If the current requested query results is *not* the first one (denoted as q_i where $i > 1$), then the algorithm will do the followings (Lines 06–10). First, it clones the `FirstFoundVRules` table into the `FoundVRules` table (Line 07) using the SQL

```
01  UPDATE FoundVRules SET COUNTER =  C.TOTAL
02  FROM  FoundVRules F,
03    (SELECT C.RULENO, C.OBJID,
              F.COUNTER - COUNT(*) AS TOTAL
04     FROM DEL_OBJ D, RULES C, FirstFoundVRules F
05     WHERE CHARINDEX(' ' + C.OBJECT + ' ',
                       ' ' + D.VALUE  +' ') >0
06       AND C.RULENO = F.RULENO
07       AND C.OBJID = F.OBJID
08     GROUP BY C.RULENO, C.OBJID, F.COUNTER) C
09  WHERE C.RULENO = F.RULENO  AND C.OBJID = F.OBJID
```

(a) Analyze the Changes

```
01  SELECT DISTINCT C.RULENO, C.CLABEL
02  FROM RULES C,
03    (SELECT F.RULENO,
             COUNT(F.OBJID) AS VOBJ
04     FROM FoundVRules F
05     WHERE F.COUNTER >0
06     GROUP BY F.RULENO) F
07  WHERE F.RULENO = C.RULENO
08    AND F.VOBJ = C.TOTALOBJECT
```

(b) Find Violated Rules

Fig. 6. SQL queries used in DIFF approach.

Dataset	Number of Leaf Nodes	Filesize (KB)
1	258	13
2	427	21
3	703	34
4	1,437	70
5	2,151	104
6	3,734	180

(a) Data Set

N	R	Total Objects
2	50	100
4	50	200
6	50	300
8	50	400
10	50	500

N	R	Total Objects
2	500	1,000
4	500	2,000
6	500	3,000
8	500	4,000
10	500	5,000

N	R	Total Objects
2	5,000	10,000
4	5,000	20,000
6	5,000	30,000
8	5,000	40,000
10	5,000	50,000

(b) Clearance Rules Characteristics

Fig. 7. Dataset and clearance rules characteristics.

query depicted in Figure 5(d). This step is important as we want to evaluate clearance for q_i using the clearance of q_1. Next, the algorithm determines the differences between q_1 and q_i by executing two SQL queries. The first SQL query is used to find the deleted objects (Figure 5(e)). Note that did_1 and did_2 will be replaced by the ids of q_1 and q_i, respectively. The result of this SQL query is stored in the `Del_Obj` table (Figure 4(b)). The second SQL query is used to detect the inserted objects. We use the same SQL query as shown in Figure 5(e); however, $did1$ and $did2$ will be replaced by the ids of q_i and q_1, respectively. The results of this SQL query is kept in the `Ins_Obj` table (Figure 4(b)).

Having found the differences between q_1 and q_i, the algorithm analyzes the deleted and inserted objects based on the Property 1 and Property 2, respectively, in order to determine the clearance of q_i. The SQL query depicted in Figure 6(a) is executed to analyze the set of deleted objects. Line 3 is used to decrease the number of appearances of sensitive objects if the sensitive objects are deleted. Similarly, this query is slightly modified to analyze the inserted objects. The modifications are as following. The "-" in Line 3 is replaced by "+". In Line 4, we replace "DEL_OBJ" with "INS_OBJ".

The last step in evaluating each requested result q_i is to find the rules that raise security cautions by querying the `FoundVRules` table (denoted by V). Figure 6(b) presents the SQL query for determining such rules. Then, we add a pair of request ids i and V into Z. Finally, the algorithm returns Z which may be analyzed further in order to determine which requested results are safe for publication.

3 Experimental Results

In this section, we present the experiments conducted to evaluate the performance of our proposed approach and report some of the results obtained. A more detailed results is available in [7]. The experiments were conducted on a computer with Pentium 4 3GHz processor and 1GB RAM. The operating system was Windows XP Professional. All the approaches were implemented using Java JDK 1.6. We use Microsoft SQL Server 2005 Developer Edition as our backend database system.

We use synthetic XML documents that are generated based on the DTD of SIGMOD Record XML. We assume that these documents represent results requested by the *Policy*

Data Set	1	N=2							1	N=10						
		2			3			Avg		2			3			Avg
		A	B	Total	A	B	Total			A	B	Total	A	B	Total	
1	1.31	0.02	0.06	0.08	0.02	0.04	0.06	**0.48**	7.79	0.05	0.12	0.17	0.05	0.10	0.15	**2.70**
2	2.05	0.02	0.06	0.08	0.02	0.05	0.06	**0.73**	12.53	0.05	0.12	0.17	0.05	0.09	0.14	**4.28**
3	3.25	0.03	0.05	0.07	0.02	0.05	0.07	**1.13**	20.39	0.06	0.12	0.17	0.06	0.10	0.16	**6.91**
4	8.39	0.03	0.06	0.08	0.03	0.05	0.08	**2.85**	41.36	0.07	0.12	0.19	0.07	0.09	0.17	**13.91**
5	12.48	0.05	0.06	0.11	0.05	0.05	0.09	**4.23**	61.71	0.08	0.12	0.20	0.08	0.09	0.18	**20.69**
6	21.52	0.07	0.05	0.13	0.07	0.05	0.12	**7.25**	106.72	0.10	0.12	0.22	0.10	0.09	0.20	**35.71**

(a) R=500

Data Set	1	N=2							1	N=10						
		2			3			Avg		2			3			Avg
		A	B	Total	A	B	Total			A	B	Total	A	B	Total	
1	15.32	0.08	0.17	0.25	0.08	0.13	0.20	**5.26**	75.86	0.30	0.55	0.86	0.30	0.33	0.63	**25.78**
2	24.99	0.08	0.17	0.25	0.08	0.11	0.20	**8.48**	123.75	0.30	0.56	0.86	0.31	0.33	0.64	**41.75**
3	40.64	0.08	0.17	0.25	0.09	0.12	0.21	**13.70**	202.21	0.31	0.56	0.87	0.30	0.34	0.64	**67.91**
4	82.63	0.10	0.16	0.26	0.10	0.12	0.23	**27.70**	412.95	0.36	0.56	0.92	0.37	0.34	0.70	**138.19**
5	123.42	0.12	0.17	0.29	0.12	0.12	0.23	**41.31**	618.45	0.38	0.57	0.94	0.38	0.33	0.72	**206.70**
6	219.92	0.13	0.17	0.29	0.14	0.12	0.26	**73.49**	1069.9	0.40	0.57	0.97	0.40	0.33	0.74	**357.20**

(b) R=5000

Fig. 8. Experimental results: The DIFF approach (in seconds).

Enforcers. Each data set has three different versions. Figure 7(a) depicts the characteristics of our data sets. The clearance rules are generated by randomly choosing the objects that co-occur together. The numbers of clearance rules (denoted as R) are between 50 and 5,000 rules, and the number of objects in each rule (denoted as N) are between 2 and 10. Hence, the total number of sensitive objects in the clearance rules is between 100 and 50,000 (Figure 7(b)).

The performance of the DIFF approach for $R = \{500, 5000\}$ and $N = \{2, 10\}$ is depicted in Figure 8. The "*A*" and "*B*" columns denote the execution times of finding the changes and of analyzing the changes, respectively. Observe that as the values of N and R increase, the performance becomes slower. The performance of analyzing the first document version is slower compared to the subsequent versions as the whole document is analyzed for the clearance rules. The performance of analyzing the subsequent versions is significantly faster as much lesser number of objects are evaluated.

4 Conclusions and Future Work

In this paper, we have presented a novel and sophisticated approach for automatically evaluating sensitiveness of publishing a batch of XML documents in a federated XML access control environment, and giving security clearance based on the sensitiveness. We use the differences between requested query results for clearance evaluation in our model. Our experimental results show that the proposed diff-based approach is efficient in determining security clearance. As part of future work, we would like to extend our framework to support clearance of security objects that are semantically related.

References

1. Liberty Alliance Project Homepage, http://www.projectliberty.org/.
2. OpenID Foundation, http://openid.net/.
3. E. COHEN, R. K. THOMAS, W. WINSBOROUGH, D. SHANDS. Models for Coalition-based Access Control. *In ACM SACMAT*, 2002.
4. G. GOU, R. CHIRKOVA. Efficiently Querying Large XML Data Repositories: A Survey. *In IEEE TKDE*, 19(10), 2007.
5. B. HAYES. Cloud Computing. *Communications of the ACM*, Vol. 51, No. 7, pp.9–11, Jul. 2008.
6. A. KERN, M. KUHLMANN, A. SCHAAD, J. MOFFETT. Observations on the Role Life-Cycle in the Context of Enterprise Security Management. *In ACM SACMAT*, 2002.
7. E. LEONARDI, S. S. BHOWMICK, M. IWAIHARA. Efficient Database-Driven Evaluation of Security Clearance for Federated Access Control of Dynamic XML Documents. *Technical Report* , Available at `www.cais.ntu.edu.sg/~assourav/TechReports/XMLSecurity-TR.pdf`, 2009.
8. D. LIN, P. RAO, E. BERTINO ET AL. Policy Decomposition for Collaborative Access Control. *In SACMAT*, 2008.
9. S. PRAKASH, S. S. BHOWMICK, S. K. MADRIA. SUCXENT: An Efficient Path–Based Approach to Store and Query XML Documents. *In DEXA*, 2004.