A Practical Mandatory Access Control Model for XML Databases

Hong Zhu[1], *Kevin Lü*[2], *Renchao Jin*[1]
[1]Huazhong University of Science and Technology, Wuhan, Hubei, 430074, P.R. China
[2]Brunel University, Uxbridge, UB8 3PH, UK

Abstract. A practical mandatory access control (MAC) model for XML databases is presented in this paper. The label type and label access policy can be defined according to the requirements of different applications. In order to preserve the integrity of data in XML databases, a constraint between a read-access rule and a write-access rule in label access policy is introduced. Rules for label assignment and propagation are presented to alleviate the workload of label assignments. Furthermore, a solution for resolving conflicts in label assignments is proposed. Rules for update-related operations, rules for exceptional privileges of ordinary users and the administrator are also proposed to preserve the security of operations in XML databases. The MAC model, we proposed in this study, has been implemented in an XML database. Test results demonstrated that our approach provides rational and scalable performance.

Keyword: mandatory access control, XML databases, access control model, security.

A Practical Mandatory Access Control Model for XML Databases

Abstract. A practical mandatory access control (MAC) model for XML databases is presented in this paper. The label type and label access policy can be defined according to the requirements of different applications. In order to preserve the integrity of data in XML databases, a constraint between a read-access rule and a write-access rule in label access policy is introduced. Rules for label assignment and propagation are presented to alleviate the workload of label assignments. Furthermore, a solution for resolving conflicts in label assignments is proposed. Rules for update-related operations, rules for exceptional privileges of ordinary users and the administrator are also proposed to preserve the security of operations in XML databases. The MAC model, we proposed in this study, has been implemented in an XML database. Test results demonstrated that our approach provides rational and scalable performance.

1 Introduction

XML has been increasingly used in a variety of applications, and has become the standard for describing and exchanging data over the Internet. As more and more XML documents are stored in XML databases, the security of XML databases has become an important issue. Access control is one of the most important measures in guaranteeing the security of XML databases. The existing access control models for XML databases can be generally catalogued into two types: the discretionary access control model (DAC) and the mandatory access control model (MAC). In the DAC models, a subject who is a creator of an object can discretionally control privileges of other subjects accessing the object in XML databases but cannot resist attacks like *Trojan horse* because of inherent flaws.

In a MAC model based on BLP model [2], every object is assigned a label which specifies the security privilege of the object, and every user is assigned a label which specifies what objects he/she can access. The label in [2] is a binary-tuple $L=<l, c>$ consisting of a classification l, which belongs to an ordered sequence (e.g. *unclassified* $\le$ *confidential* $\le$ *secret* $\le$ *top-secret*); and a category c, which is a subset of a set consisting of non-hierarchical and unordered elements (e.g.{*HumanResource*, *Financial, Technical*}). In the following discussion, we assume that $L(s)$ and $L(o)$ denote the labels for a subject and an object respectively. $L(s).l$ is the classification of the label $L(s)$ and $L(s).c$ represents the category for the subject s. When an object is accessed by a subject, the label of the subject is compared with the label of the object using the following two rules:

(1) simple security property: A subject s can read an object o only if $L(o) \le L(s)$.

*(2) *-property*: A subject s can write an object o only if $L(s) \le L(o)$.

In the *simple security property*, label $L(o) \le L(s)$ if and only if $L(o).l \le L(s).l$ and $L(o).c \sqsubseteq L(s).c$. We call $L(o) \le L(s)$ as $L(s)$ dominates $L(o)$. However, when applying the BLP model to databases, the **-property* has to be modified to maintain the integrity of the data. Writing objects is allowed only when the subjects and objects have the same label. Therefore, the *simple security property* and the *strict *- property* need to be enforced [1, 15]. The *strict *-property* is described as follows:

*strict *- property*: A subject s can write an object o only if $L(o)=L(s)$.

The MAC security of a system based on the BLP model is sufficient in many cases, and it has been widely applied in military and government information systems [2]. However, the rules for label comparison in these systems are too rigorous in some cases. It indicates that the larger the category for an object is, the fewer the users that can access it. In some applications [18, 21], the rule for label comparison is not as rigorous as this**.** On the contrary, the requirement is: the larger the category for an object is, the more the users that can access it. Moreover, in these applications the structure of the label may be different from the structure of the label in the BLP model. In order to meet the needs of these applications, [14, 18, 21] have enhanced the flexibility of the MAC mechanism in relational databases. However, the existing MAC models [4, 15, 23] for XML databases are all based on the BLP model. The issue of how to make a flexible and practical MAC model for XML databases has not been addressed in the literatures. In order to overcome the limitations of previous MAC models for XML databases, new rules have to be introduced to make a MAC model be more flexible and can be applied for multi-purposes, meanwhile the security properties of the MAC model are maintained. That is the focus of this study.

We propose a mandatory access control (MAC) model that consists of 12 rules specifying the constraints between a read-access rule and a write-access rule in label access policy, label assignment and propagation, a solution for resolving label assignment conflicts, and updating-related operations (including *INSERT*, *UPDATE* and *DELETE*) for XML databases. Fig. 1 illustrates the relationships amongst these 12 rules in our MAC model. Every node denotes a rule and the edge between two nodes denotes the relationship between the two rules. If there is an edge from *rule A* to *rule B*, *rule A* must take effect prior to *rule B*. After the structure of the labels and the label access policy are defined in the MAC, *Rule 1* specifies the constraint between the read-access rule and the write-access rule in label access policy to preserve the integrity of data in XML databases. Moreover, several label access policies can be defined in an XML database. Different label access policies can be assigned to different XML schemas and XML documents, but one document can only be assigned with one policy. *Rule 2* specifies that a XML schema and its documents must be assigned with the same label access policy, and then a subject and an object can be assigned a label respectively. *Rule 3* specifies that the label of the root of a schema is the label of its creator. Only after the XML schema is loaded into an XML database, XML documents can then be loaded. *Rule 7* indicates that how to specify the label of the root of a XML document that has been loaded. There are three methods of assigning labels to objects in an XML document: assign labels to objects arbitrarily; propagate labels from the ancestors of the objects; and propagate labels from the nodes in a schema associated with the objects. *Rule 4* and *Rule 5* are related to label propagation. *Rule 6* is for deciding the unique label for every object in an XML document. After every element or attribute in an XML document is assigned a label, the update-related operations for the document can be applied, which are specified in *Rule 8, Rule 9* and *Rule 10.* Moreover, the exceptional privileges for ordinary users who create the XML documents and the exceptional privileges for administrator are specified in *Rule 11* and *Rule 12*.

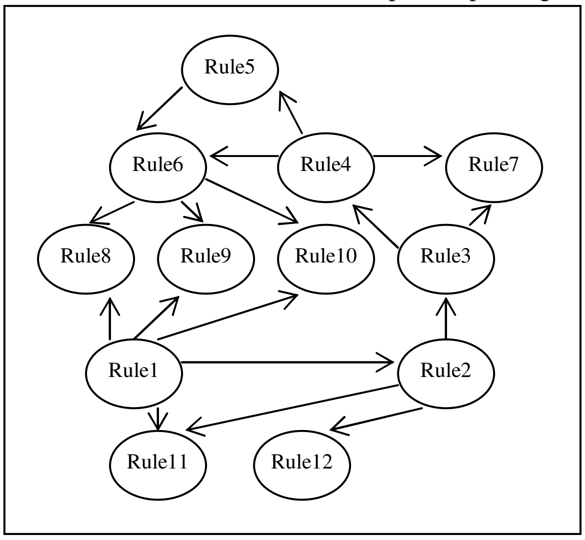

Fig.1. The relationships of rules in our model

```
<LabelType name = "COMDEPT">
      <LabelComponents>
        <LabelComponent name = "Secret" type = "order">
          <value>unclassified </value>
          <value>secret</value>
          <value>top-secret</value>
      </LabelComponent>
     <LabelComponent name = "Dept" type = "unorder">
          <value>Technique</value>
          <value>HumanResource </value>
          <value>Financial</value>
   </LabelComponent>
  </LabelComponents>
</LabelType>
```

Fig. 2. An example for the label type

The main contributions of this paper are:

(1) A practical MAC model for XML databases is proposed. The label structure and label access policy can be defined according to the application requirements. We not only can define label access policies which are similar to label access policies in the BLP model for applications with high security, but also can define label access policies according to the application requirements in the way which is different from the BLP model. Therefore, our practical MAC model for XML databases can be used for more different applications.

(2) In order to preserve the integrity of data in XML databases, a constraint between the read-access rule and write-access rule in label access policy has been introduced.

(3) The rules for label assignment and label propagation are addressed to alleviate the workload of label assignments. A solution for the problem of conflicts in label assignments has been proposed.

(4) The rules for update-related operations and the rules for exceptional privileges of both ordinary users and the administrators are developed to preserve the security of operations involving *SELECT*, *UPDATE*, and *DELETE* in XML databases.

(5) We implemented our newly proposed practical MAC model in an XML database. All newly proposed rules and functions have been implemented. This enables us to verify the feasibility, effectiveness and flexibility of our approach. Our approach has been validated against the XMARK benchmark and the test results shown that the performance is rational and scalable.

The rest of this paper is organized as follows: Section 2 illustrates the preliminary concepts of our model. Section 3 describes the practical MAC model for XML databases in detail. Section 4 discusses the implementation considerations of the system. Section 5 illustrates the experimental results. Section 6 describes the related work. Finally, Section 7 presents the conclusions and suggestions for future work.

2 Preliminary Concepts

The structure of the label in our MAC model is as follows: each label is specified by a label type, and the label type specifies what components the label type consists of, i.e. every component in a label is specified by a label component type. Normally, the structure of a label is specified by the administrator, which may or may not be the same as the structure of the label in the BLP model.

Definition 1. Label component type. The label component type LC is a set of elements $LC=\{c_1,c_2,\ldots,c_m\}$, $m\geq 1$ and it also specifies whether the set is ordered or unordered. If LC is an ordered set, then $c_1 \leq c_2 \leq \ldots \leq c_m$ and we call it an ordered label component type. Otherwise, we call it an unordered label component type.

Definition 2. Label component. The label component lc is an instance of a label component type. If LC is an ordered label component type then $lc \in LC$, otherwise $lc \subseteq LC$.

Example 1. As shown in Fig.2, two label component types *Secret* and *Dept* are defined, where *Secret*=LC_1=*{unclassified, secret, top-secret}* is an ordered label component type with *unclassified* ≤ *secret* ≤ *top-secret*. Any element in LC_1 is a label component similar to the classification of the label in the BLP model; *Dept*=LC_2=*{Technique, HumanResource, Financial}* is an unordered label component type, *Dept* is a set of the department names of a company. Any subset of LC_2 is a label component similar to the category of the label in the BLP model.

Definition 3. Label type. A label type is an n-tuple $LT=<LC_1, \ldots, LC_n>$, which is composed of n label component types. Here $n\geq 1$, LC_i is a defined label component type. In order to prevent semantic confusion amongst labels, we specify that there is at most one ordered label component type in a label type. If a label type has an ordered label component type, then we specify that LC_1 must be an ordered label component type.

Example 2. Using a label type specifies the structure of the label. In Fig.2, a label type *COMDEPT=<Secret, Dept>* is defined.

Definition 4. Label. A label $L = <lc_1, ..., lc_n>$ is an instance of a label type $LT =< LC_1, LC_2, ..., LC_n >$. Where LC_i ($1 \le i \le n$) is a defined label component type, and lc_i is a label component of LC_i.

Unlike the label in the BLP model, more than one unordered component type in a label type can be defined in our MAC model according to the various application requirements.

Definition 5. Label access rule. The label access rule is a four-tuple *LAS=(L, T, OP, type),* where *L* is a set of labels and *T* is the label type of *L*, $type \in \{r, w\}$ indicates the rule is a read-access rule (*type=r*) or a write-access rule (*type=w*), and *OP* is a n-tuple $OP=<op_1, ..., op_n>$. For any two labels l_1, $l_2 \in L$, each op_i indicates the operator between $l_1.c_i$ and $l_2.c_i (1\le i \le n)$. For an ordered label component type in *T*, the $op_i \in \{EQ, LE, GE, GT, LT\}$ denotes operators in *{=, <=, >=, >, <,}* respectively. For an unordered label component type in *T*, the $op_i \in$ *{IN, INTERSECTION, CONTAIN, EQUAL}.*

Definition 6. Label access policy. The label access policy is a triple *LP=(S, O, LASS),* where *S* is a set of subjects, *O* is a set of objects, and *LASS* is a set of label access rules. The *LASS* consists of a read-access rule and a write-access rule.

LP indicates that when a subject $s \in S$ reads an object $o \in O$, the labels for the subject and the object must conform to the read-access rule. The labels for the subject and the object are compared according to the operations specified in *LP.LASS.OP* to determine whether the subject can read the object or not. In the following discussion, if a label *L(s)* and a label *L(o)* conform to a read-access rule in *LP.LASS*, it is denoted as $L(s)\omega_{r\ in\ LP.LASS}\ L(o)$, if a label *L(s)* and a label *L(o)* conform to a write-access rule in *LP.LASS*, it is denoted as $L(s)\omega_{w\ in\ LP.LASS}\ L(o)$.

For a label access policy, we can define label access rules in an XML format file. The format of a rule for specifying an operator between two components in the labels of a subject and an object is:

Subject.component-name <operator> Object.component-name

Example 3. Defining a label access policy. In Fig.3, a label access policy for the label type *COMDEPT* (in Fig.2) is defined. It consists of a read-access rule and a write-access rule.

```
< Policy labeltype = " COMDEPT ">
<!-- A subject is allowed to read an object, iff Secret component of the subject is higher than or equal to --!>
<!--Secret component of the object and the intersection of Dept of the subject and Dept of the object are not empty --!>
<!-- A subject is allowed to write an object, iff Secret component of the subject is less than or equal to --!>
<!--Secret component of the object and Dept component of the subject is in Dept component of the object -->
    <Rules action = "read" >
        <Rule> subject.Secret GE object.Secret </Rule>
        <Rule> subject.dept  INTERSECTION  object.dept </Rule>
    </Rules>
    <Rules action = "write" >
        <Rule>subject.Secret LE object.Secret</Rule>
        <Rule>subject.dept  IN  object.dept</Rule>
    </Rules>
</Policy>
```

Fig. 3. A label access policy

In XML databases with our mandatory access control, the labels of a subject and an object are compared according to the label access rules. Our model supports the *INTERSECTION* operator when we compare two labels; this is different from the other MAC models [4, 15, 23] for XML databases. However, when the operation

INTERSECTION appears in the read-access rule, the domination relationship between two labels has not been defined in our MAC model; therefore, less restriction has been imposed to make it more flexible. This can be demonstrated in the following example.

Example 4. Assume L_1=*<secret, {Technique, Financial}>* and L_2=*<secret, {HumanResource, Financial}>* are two labels, L_1 is assigned to a user and L_2 is assigned to an object. Clearly, the user could read the object under the label access policy shown in Fig.3 because L_1*.secret*≥L_2*.secret* and L_1*.{Technique, Financial} INTERSECTION* L_2*.{HumanResource, Financial}* is not empty (L_1 and L_2 matched the read-access rule), meanwhile L_2*.secret*≥L_1*.secret* and L_2*.{Technique, Financial} INTERSECTION* L_1*.{HumanResource, Financial}* is neither not empty (L_2 and L_1 also matched the read-access rule). However, in the BLP model, if L_1 and L_2 as well as L_2 and L_1 matched the *simple security property*, then $L_1=L_2$. Therefore, under the label access policy in Fig.3, for such a case, whether L_1 dominates L_2 or L_2 dominates L_1 can not be decided. But in our model, the relationship between L_1 and L_2 can still exist.

3 The Mandatory Access Control Model

3.1 The rule for label access policy

When a subject attempts to access an object, the read-access rule is applied to decide whether the user can read the object. When a subject attempts to insert an object into an XML document, or attempts to update or delete an object from an XML document, the read-access rule must be evaluated to locate the object to be inserted, updated or deleted. Therefore, in our MAC model any user can read what he/she wrote before. Otherwise, the integrity of data would not be maintained. Therefore, we introduce the following constraint:

Rule 1. The constraint of label access policy. *For a label access policy LP=(S, O, LASS), the write-access rule contains the read-access rule. Namely, for any subject s∈ S, and any object o∈ O, the privilege of L(s)* $\omega_{w \text{ in } LP.LASS}$ *L(o) is higher than L(s)* $\omega_{r \text{ in } LP.LASS}$ *L(o).*

3.2 Labelled subjects and XML documents

The subjects of our MAC model are the users who access the XML databases, or application programs, or agents on behalf of users. After the label type is created, the administrator assigns labels to users, elements or attributes in XML documents or schemas. Every user except the administrator has a unique label, and so does as every object in XML documents.

The objects are elements/attributes in XML documents and schemas. We denote the objects in XML documents using XPath [19]. As the XML document and schema conform to the same XML grammar, the labelled schema and its XML documents have the same notations. We only formally describe the labelled XML document below. The same principle can be applied for formally defining XML schema.

Definition 7. Labelled XML document. The XML document *XDoc* with labels is an eleven-tuple *XDoc=(Ve, vr, Va, Ns, Ls, T, LP, elemR, attrR, nameR, labelR)*. We have:

(1) *Ve* is the set of all elements in the document;

(2) *vr* is the root of the document, *vr* is also an element of the document, *vr*∈ *Ve*;

(3) *Va* is the set of all attributes in the document;

(4) *Ns* is the set of names, including the names of elements and attributes;

(5) *Ls* is the set of all labels with a label type *T*;

(6) *T* is the label type which specifies the structure of the labels in the document;

(7) *LP* is the label access policy, and *LP* determines the set of label access rules including a read-access rule and a write-access rule;

(8) *elemR* is a binary-tuple, $elemR \subseteq Ve \times Ve$. If $e1 \in Ve$, $e2 \in Ve$, then $(e1,e2) \in elemR$ denotes *e2* is a sub-element of *e1* or there exists a link between *e1* and *e2*;

(9) *attrR* is a binary-tuple, $attrR \subseteq Ve \times Va$. If $e \in Ve$, $a \in Va$ then $(e, a) \in attrR$ denotes *a* is an attribute of *e*;

(10) *nameR* is a binary-tuple, $nameR \subseteq Ns \times (Va \cup Ve)$. If $n \in Ns$, $v \in Va \cup Ve$, then $(n, v) \in nameR$ denotes that *n* is the name of *v*. As different elements or attributes in the same document may have the same names, one member of *Ns* may be mapped into different members of $Va \cup Ve$;

(11) *labelR* is a binary-tuple, $labelR \subseteq (Va \cup Ve) \times Ls$. If $L \in Ls$, $v \in Va \cup Ve$, then $(v, L) \in labelR$ denotes *L* is the label of *v*, or $L=L(v)$. Different elements or attributes may have the same label, and every element or attribute has only one label.

For a document not being labelled, it is a seven-tuple *Doc=(Ve, vr, Va, Ns, elemR, attrR, nameR)*. The meanings of these symbols are the same as those in the *XDoc*.

In the following discussion, we use *XDoc* and *XSch* to denote a labelled XML document and schema, respectively. For example, we also denote *XDoc.T* and *XDoc.LP* as a label type and a label access policy for the XML document. From the definition of *XDoc*, we have: for any label access rule $las \in XDoc.LP.LASS$, $XDoc.T=las.T$.

3.3 The label assignment rules for XML objects

Multiple label access policies can be defined for different security requirements but every document can be assigned only one label access policy. After an XML document or schema is loaded, a label access policy is first assigned for them. Then, the labels for elements or attributes in the XML document and schema are assigned.

3.3.1 The constraint of a label access policy between an XML schema and its documents

An XML schema defines a set of XML documents with the same structures and similar contents. The label access policy should be the same for the schema and its documents.

Rule 2. The constraint of label access policy between an XML schema and its documents*. If the labelled XML schema is XSch and one of its documents is XDoc, then XSch.LP=XDoc.LP.*

3.3.2 The rules for label assignment and propagation

When an XML schema is created, the label for the root of the schema should be assigned. For an ordinary user, when he/she creates an XML schema, the label for the root of the schema is equal to the label of the user. So we have the following *Rule 3:*

Rule 3. The label for the root of an XML schema. *For an ordinary subject s, if s creates an XML schema sch, then L(sch.vr)=L(s).*

If the schema is created by the administrator, the label for the root of the schema must be assigned explicitly. *Rule 12* (in section 3.3.5) is about the exceptional privilege for the administrator.

There are a large number of elements and attributes in an XML document. If each element or attribute is assigned a label, the workload for management of these labels would be enormous. One solution to his is that we may make use of the following features of XML to alleviate the administrator's workload:

(1) An XML schema defines a set of XML documents with the same structure and similar contents. Usually, these XML documents have the same security attributes;

(2) XML documents are hierarchical and every element has its sub-elements or attributes except for the leaves.

The administrator needs to only assign labels to XML schemas and some elements in XML documents, then the labels are propagated to instance elements or attributes in the XML documents of the schema, or the labels are propagated to descendent elements and attributes of the labelled elements downward from the root to leaves.

Rule 4. Label propagation from an XML schema to XML documents. *For a labelled XML document XDoc and its schema XSch, an instance object **io**$\in$ XDoc.Va$\bigcup$XDoc.Ve, and a schema object **so**$\in$ XSch.Va$\bigcup$XSch.Ve, assume **io** is one of the instances of the schema object **so**, then L(**io**)=L(**so**).*

Rule 5. Label propagation from an element to its sub-elements and attributes. *Assume XDoc is a labelled XML document, for any element e1, e2$\in$ Ve (or attribute a1$\in$ Va), if (e1, e2)$\in$ elemR (or (e1, a1)$\in$ attrR), then L(e2)=L(e1)(or L(a1)=L(e1)).*

3.3.3 A solution for conflicts of label assignments

Rule 3, *Rule 4* and *Rule 5* are designed to enhance the flexibility of label assignment and alleviate the workload of the administrator, but may result in the assignments of several different labels to one object. For example, an element in an XML document may have three labels. The first one is propagated from its ancestor, the second one is propagated from the schema, and the third one is assigned directly by the administrator. In order to guarantee that every object in an XML document has only one label, we introduce a rule for deciding labels under a specific label access policy to solve label assignments conflicts.

Rule 6. The label deciding rule. *Assume a read access rule lasr=(L, T, OP, r), L_1 =<a_1, a_2, …, a_n>$\in$ L and L_2 =< b_1, b_2, …, b_n >$\in$ L are two labels of label type T, if L_1 and L_2 are two labels assigned to the same object by direct assignment or by propagation, respectively, we can decide and assign a new label L_3=<c_1, c_2, …, c_n> for the object, where each component of L_3 is decided as follows:*

(1) For the ordered label component type in the T, if the operator in the lasr.OP.op_1 is:

(i) GE or GT, then $L_3.c_1=max(L_1.a_1, L_2.b_1)$;

(ii) LE or LT, then $L_3.c_1=min(L_1.a_1, L_2.b_1)$;

(iii)EQ, then $L_3.c_1=max(L_1.a_1, L_2.b_1)$;

Here the max(l_1, l_2) is a function to decide the maximum of l_1 and l_2, min(l_1, l_2) is a function to decide the minimum of l_1 and l_2.

(2)In the following formulas, if there is an ordered label component type in the label type then i≥2, otherwise i≥1. For an unordered label component type in T, if the operator lasr.OP.op_i is:

(i) IN, then $L_3.c_i = L_1.c_i \bigcap L_2.c_i$;

(ii) CONTAIN, then $L_3.c_i = L_1.c_i \bigcup L_2.c_i$;

(iii) INTERSECTION, then $L_3.c_i = L_1.c_i \bigcap L_2.c_i$;

(iv) EQUAL, then $L_3.c_i = L_1.c_i$.

Rule 6 can be extended to decide and assign a unique label for an object when the object may be assigned up to three labels due to direct assignment or propagation. Because the *least upper bound* does not exist for the *INTERSECTION* operator in the sense of the *least upper bound* defined for the *CONTAIN* operator in [15], a simple deciding rule for an *INTERSECTION* operator is used in *Rule 6*. When an *INTERSECTION* operator is specified for a component in a label access policy, we specify that the component of the resulting label is at the intersection of the corresponding components of two labels. Such a case may occur when some object cannot be accessed by any users except the administrator or the creator of the XML document which the object belongs to. The example below illustrates this problem.

Example 5. If the labels <*secret, {Technique, Financial}*> and <*secret, {HumanResource}*> are assigned to an object by propagation rules and direct assignment, and the intersection of *{Technique, Financial}* and *{HumanResource}* is empty, then the object cannot be accessed by any user except the administrator and the creator of the XML document which the object belongs to.

Consequently, *Rule 6* guarantees that the security of a sub-node is higher than the security of its parent in an XML document. In the following discussion, we use function *label_comput*(L_1, L_2) to denote the result from *Rule 6* where L_1, L_2 are two labels. Based on *Rule 1* to *Rule 6*, we have the following property:

Property: *For a labelled XML document XDoc and its label access policy XDoc.LP, for elements e1, e2*$\in$ *Ve (or attribute a2*$\in$ *Va), and an ordinary user s, if (e1, e2)*$\in$ *elemR (or (e1, a1)*$\in$ *attrR), from Rule1 ~ Rule 6, we have:* *(1) if* $L(s)\ \omega_{r\ in\ XDoc.LP.LASS}\ L(e2)$ *(or* $L(s)\ \omega_{r\ in\ XDoc.LP.LASS}\ L(a1)$*) then* $L(s)\omega_{r\ in\ XDoc.LP.LASS}\ L(e1)$*.*

(2) if $L(s)\ \omega_{w\ in\ XDoc.LP.LASS}\ L(e2)$ *(or* $L(s)\omega_{w\ in\ XDoc.LP.LASS}\ L(a1)$*) then* $L(s)\ \omega_{w\ in\ XDoc.LP.LASS}\ L(e1)$*.*

3.3.4 The rules for modification-related operations

As the label access policy for an XML database is defined according to the security requirements of the applications, modification-related operations may have an impact on the labels or the structure of XML documents. When a subject loads an XML document, the label of the subject is compared with the label of the root of the schema. If they conform to the write-access rule in the label access policy, the label of the root would be decided from the label of the subject and the label of the root of the schema of the loaded XML documents by *Rule 6*. We have the following *Rule 7* for loading an XML document:

Rule 7. Loading XML documents. Assume *XSch* is a labelled XML schema, *s* is a subject, if $L(s)\omega_{w\ in\ XDoc.LP.LASS}$ *L(XSch.vr)* and *s* loads an XML document *Doc* conforming to *XSch*, then the loading would be successful and *L(Doc.vr)= label_comput(L(s), L(XSch.vr))*.

Rule 6 and *Rule 7* indicate that the MAC can resolve the security problems in the DAC models. We give the following example.

Example 6. Assume that we have only enforced the DAC in an XML database. A user *s1* could access several objects (not all objects) in an XML document *Doc1*, but user *s2* could not access any objects in *Doc1*. User *s1* could leak information in *Doc1* by creating a new document *Doc2* which consists of the objects user *s1* could access, and then user *s1* authorizes all the access privileges of all objects in *Doc2* to *s2*. Therefore, *s2* would indirectly be able to obtain the information in *Doc1*. If the MAC is enforced by the label access policy in Fig.3 in the XML database, *s1* and *s2* are assigned labels *L(s1)* and *L(s2)* respectively ($L(s1)\neq L(s2)$), so that $L(s1)\ \omega_{r\ in\ XDoc.LP.LASS}\ L(s2)$ is true and $L(s2)\ \omega_{r\ in\ XDoc.LP.LASS}\ L(s1)$ is not true. According to *Rule 7*, the label of the root in *Doc2* would be *L(s1)* (assume that there is no schema for *Doc2*). Although *s1* could authorize all the access privileges for objects in *Doc2* to *s2*, according to *Rule 6* and *Rule 7*, all of the labels for objects in *Doc2* are equal to *L(s1)*. So *s2* could not access *Doc2* because $L(s2)\ \omega_{r\ in\ XDoc.LP.LASS}\ L(s1)$ would not be true.

If a subject modifies the labels of some objects in an XML document in the update operations, some other users' access privileges may be changed unintentionally. That may cause illegal information transmission (information leaking). So we have the following *Rule 8*.

Rule 8. Label of the object for update operations. *Assume a subject s updates an object o*$\in$ *Ve*$\cup$*Va in a labelled XDoc, then L(o) is not changed.*

For insertion operations, we should first locate the object o_1 which is the parent of the object o_2 to be inserted. From the *Property*, we can also gain access to the parent of o_2. According to *Rule 4* and *Rule 5*, the inserted object o_2 may have two labels propagated from its ancestor and schema, respectively. We shall decide $L(o_2)$ and then compare the label of the user with $L(o_2)$. If the label of the user and $L(o_2)$ satisfy the write-access rule in the label access policy, the insertion operation is permitted.

Rule 9. Label of the object for insertion operations. *Assume user s is going to insert an object o_2 as a sub-node of an object o_1 in a labelled XDoc; $o_1 \in$ XDoc.Ve and $o_2 \in$ XDoc.Ve $\cup$ XDoc.Va. Assume $L'(o_2)$ is the label of o_1 propagated from the nodes in the schema of XDoc associated with o_1, $L(o_2') =$ label_comput($L(o_1)$, $L'(o_2)$). If $L(s)$ ω_w $_{in\ XDoc.LP.LASS}$ $L(o_2')$, then an insertion operation is permitted, and the final label for o_2: $L(o_2)$=label_comput($L(s)$, $L(o_2')$).*

As the insertion and deletion operations may modify the structure of the XML documents, the structure of the updated documents must conform to the constraints of their schema. When an XML schema is defined, the maximum or minimum occurrences for the sub-elements of an element may be defined. So the operations should follow the constraints: the number of sub-elements must be no more than the maximum occurrences defined for the element after the sub-element is inserted into an element; the number of sub-elements left must be no less than the minimum occurrences defined for the element after a sub-element is deleted from the element.

Rule 10. Constraints for insertion and deletion operations. *XSch is an XML schema, XDoc is one of the documents of XSch, $(e_1, e_2) \in$ XSch.elemR. Assume $(ne, e_2) \in$ XDoc.nameR, and there exists the following constraints on e_1:*

(1)The maximum occurrence of ne is max_1;

(2)The minimum occurrence of ne is min_1;

Then, after any insertion operation on e1, the occurrence of ne $\leq max_1$; after any deletion operation on e_1, the occurrence of ne $\geq min_1$.

3.3.5 Exceptional privileges

Ordinary users can load XML documents and schemas. After an ordinary user has loaded an XML document or a schema and the administrator has assigned the labels, the ordinary user may not be able to query all the objects in the document or schema because of the label access policy. In order to prevent confusions for ordinary users "missing" objects he/she loaded earlier and to prevent transmission of information covertly, we intgioduce exceptional privileges to ordinary users who create XML documents or schemas.

Rule 11. Exceptional privileges for creators of XML documents and schemas. *If an ordinary user s loads an XML document or a schema, and the administrator assigns labels to the objects in the document or the schema, then for any object o $\in$ Ve $\cup$ Va in XDoc, s can access o without complying with the label access policy.*

Although the administrator maintains labels in the system, the administrator does not have a label. How do we deal with the initial labels for the XML documents and schemas loaded by the administrator? To solve the problem, we give the following *Rule 12*.

Rule 12. Exceptional privileges for the administrator. *The administrator can load XML documents and schemas, and the administrator must also assign an initial label to the root of the loaded XML documents and schemas.*

4. Implementation Considerations

4.1 The architecture of the MAC for XML databases

We have implemented our MAC model in an XML database management system. We used Dom4J as our parser for the XML schemas and documents. The main tool for the program is JDK1.4+Eclipse3.0+Dom4J [6, 7]. The architecture of the XML database management system is shown in Fig.4. It consists of five main modules, which are all within the rectangle with dashed line: *policy management module*, *schema and document loader module*, *policy* *service module*, *query module* (process read-only operations) and *modification module* (process update-related operations).

A tool was developed for managing label access policy at the client end. With this tool, the label type and label access policy can be defined and managed. After the *policy management module* accepts requests from this tool, it converts the label access policy and label type into an XML file and stores the XML file into the security repository. There are two types of XML files in the security repository: label access policy file and label assignment file for users. The security repository is invisible to ordinary users but is visible to the administrator.

The administrator and ordinary users can load the schemas or XML documents by using the *schema and document loader module*. The initial labels for the schemas and documents are assigned by *the policy service module.* The labelled XML schemas and documents are stored into the labelled XML documents and schemas repository. Moreover, by using this tool labels are assigned to users and objects in XML documents and schemas. For objects in an XML document, the unique labels are computed and the labelled XML documents and schemas are stored.

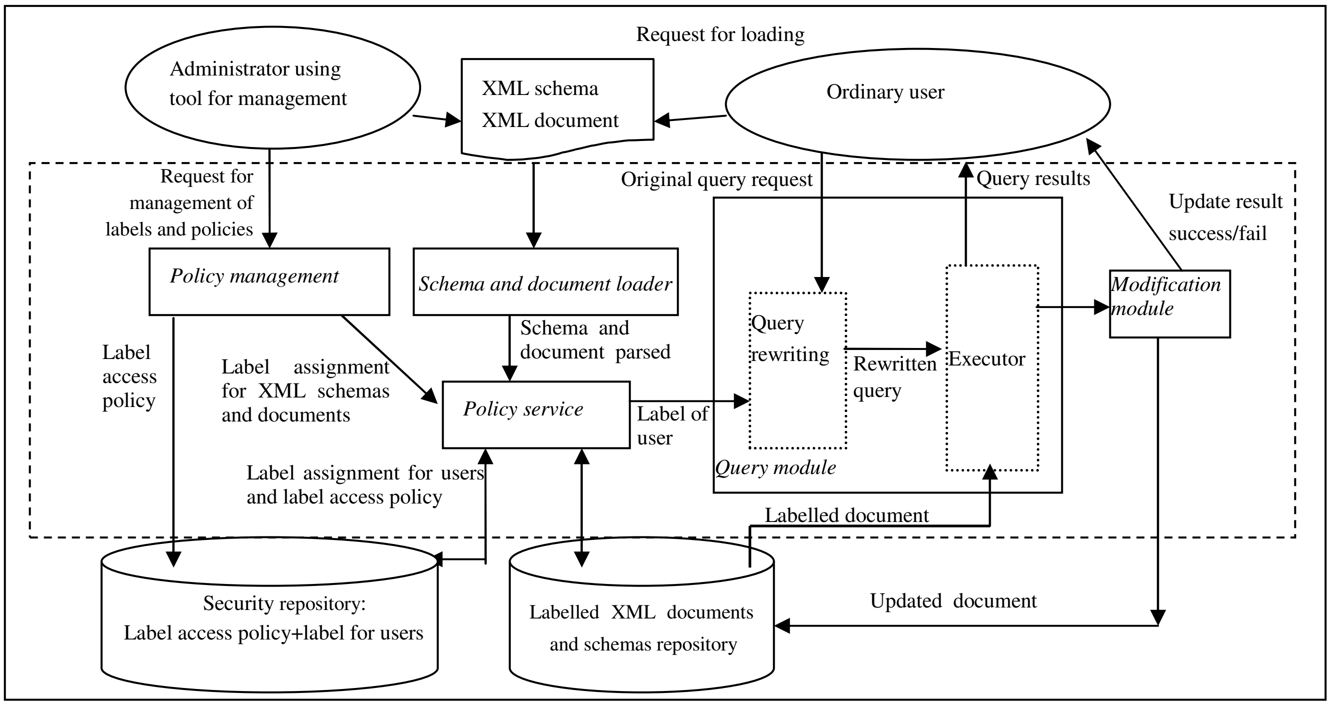
Request for loading
Administrator using tool for management
XML schema
XML document
Ordinary user
Request for management of labels and policies
Original query request
Query results
Update result success/fail
Policy management
Schema and document loader
Modification module
Query rewriting
Rewritten query
Executor
Label access policy
Label assignment for XML schemas and documents
Schema and document parsed
Policy service
Label of user
Query module
Label assignment for users and label access policy
Labelled document
Security repository: Label access policy+label for users
Labelled XML documents and schemas repository
Updated document

Fig. 4. Architecture of the MAC in XML database

The *query module* has two sub-modules: *query rewriting and executor.* When a user submits a query, in the *query module* the original query is first rewritten according to the label of the user, then the rewritten query is executed, and the final results are returned to the user. The objects in XML documents are updated by the *modification module*. When a user submits a modification request, the *query module* is executed first to find or locate the objects to be updated, deleted or inserted, and then updated documents are stored in the XML database.

4.2 Key techniques for implementation

4.2.1 The assignment of labels for users and objects in XML documents

When a label type is defined, it is converted into an inner denotation in the system. For example, an element in the order label components is denoted as a number, and a set of the unordered label components is represented as a string of *'1'* or *'0'* that denotes whether an element occurs or not. Therefore, if an operator for an unordered label component includes *CONTAIN*, and when the unordered label component for a user contains the unordered label component for an object, each *'0'* in the string for the unordered label component of the object indicates that the character in the corresponding position of the string for the unordered label component of the user should also be *'0'*. Similar processes are also developed for *IN* and *INTERSECTION* operators. By this way, labels for the users and objects are compared.

In order to assign labels to documents in an XML database, the structure of XML documents is modified. A label is attached as an attribute for each object in the documents. However, the schema of the XML documents should not be changed because the XML schema defines the structure of many XML documents. So when an XML schema is loaded, a schema map document is created for the schema. Initially, the schema map document is the XML schema, then the mapping relationships between the objects in the schema and objects in the XML documents are added, and the labels for the schema are also stored as attributes of the objects in the schema map document. The schema map document is also stored in the labelled XML documents and schemas repository. When the administrator assigns a label to the objects in XML documents or the schemas, *Algorithm 1* and *Algorithm 2* (shown in Fig.5) are executed to compute the labels for objects in XML documents.

```
Algorithm 1: Assigning a label to an object in an XML schema
Input: a parsed schema map document tree for the schema, assume Rs is the root;
       the path expression for the schema path and
       the label label1
Output: the newly labelled XML schema map document
Step1: find the object so in the schema map document tree for the path
Step2: if  (the parent of so exists)  then
           find the parent pa of so;
           if  (the label of pa exists)  then
               obtain the label paLabel of pa;
               label1 = label_comput(label1, paLabel);
           end
       end
step 3: if (so has a label) then
               obtain the label label_old of so;
               label1 = label_comput(label1, label_old);
        end
        assign label1 to so
        obtain the set of subnodes of so Set_So;
        if (the Set_So is empty) then return;
        for each c in Set_So call algorithm 1 recursively with the parameters: Rs, c, label1
end Algorithm1

Algorithm 2: Assigning a label to an object in an XML document
Input:  a parsed XML document doc1, assume Rt is the root;
        the path expression path for an object in the XML document doc1;
        the label label1
Output: the newly labelled XML document tree
Step1:  according to the path, find the objects to a set path_Set from the document tree doc1
        for each element io in path_Set do
Step2:      if  (io has a corresponding schema object)  then
                obtain the label schemaLabel for the corresponding schema object of io;
                label1 = label_comput(label1, schemaLabel);
            end
Step3:      if  (the parent of io exists)  then
                obtain the label parentLabel of the parent of io;
                label1 = label_comput(label1, paLabel);
            end
Step4:      if (io has a label label_old) then
                label1 = label_comput(label1, label_old);
            end
            assign label1 to io
            obtain the set of subnodes Set_So of io.
            if (the Set_So is empty) then return;
            for each c in Set_So call algorithm 2 recursively with the parameters: Rt, c, label1.
        endfor
end Algorithm2
```

Fig 5. Algorithms for label assignment

When all the objects in an XML document are assigned a label, the XML document is stored to the labelled XML documents and schemas repository. Clearly, the size of an XML document would be increased and that would affect the performance of the system because of the extra I/O and other operations costs.

4.2.2 Query rewriting

After the users and XML documents are labelled, the system could accept the *Query* operation (deals with read-only operations - *SELECT* operations) and update-involved operations in the XML database. Because the *Query* operation is an important operation in the system and is the basis of other operations, we describe the process of *Query* operation in the system in detail.

```
CreatePredicate: generate security predicate according to the label of the user and the read-access control rule
input:    label of user userLabel
          the read-access control rule AccessRule
output:   the security predicate for the query: predicate
          predicate = "";  LABEL = "@label";
          obtain the number of components m from AccessRule
          for i = 1 to m do
               begin
                    obtain the ith operation $p_i$ in AccessRule.OP.$p_i$;
                    sub_predicate = "";
                    obtain the ith component coml of userLabel;
                    switch ($p_i$)
                         case CONTAIN:  for each character $ch_j$ in coml do
                                        if ($ch_j$= = '0') then
                                             if (sub_predicate = ="") then sub_predicate = "substring(" + LABEL + i+ "," + j + "," +'0');
                                               else sub_predicate += " AND substring(" + LABEL + i+ " , " + j + ", " +'0');
                                        endif
                                        break;
                         case INTERSECTION:
                                        for each character $ch_j$ in coml do
                                        if ($ch_j$ = = '1') then
                                             if (sub_predicate = ="") then sub_predicate = " substring(" + LABEL + i +"," + j + "," +'1');
                                              else sub_predicate += " OR substring(" + LABEL + i +"," + j + "," +'1');
                                        endif
                                        break
                         case  IN:     for each character $ch_j$ in coml do
                                        if ($ch_j$ = = '1') then
                                             if (sub_predicate = ="") then sub_predicate = " substring(" + LABEL + i +", " + j + "," +'1');
                                                       sub_predicate += " AND substring(" + LABEL + i+"," + j + "," +'1');
                                                endif
                                              break;
                          case EQUAL:  sub_predicate = LABEL + i + "=" + coml;   break;
                         case >=: sub_predicate = LABEL + i + "<=" + coml;    break;
                         case >:   sub_predicate = LABEL + i + "<" + coml;      break;
                         case <=: sub_predicate = LABEL + i + ">=" + coml;   break;
                         case <:   sub_predicate = LABEL + i + ">" + coml;     break;
                    endswitch
                   if (predicate=="") then predicate = sub_predicate else  predicate += " AND (" + sub_predicate + ")";
               enddo
           return   predicate;
endCreatePredicate

Query_rewritten: rewrite the query Q according to the label of the user userLabel
Input:    the query Q expressed by Xpath;
          the label of the user userLabel
Output: the rewritten query Qd
          obtain the labelled schema sch for query Q;
          obtain the read-access control rules AccessRule according to sch;
          predicate = CreatePredicate(userLabel, AccessRule,);
          decompose the query Q into a set of searching expressions CC according to the separator '/' or '//' or other separators;
          /* if '/' or '//' is included in an expression, we did not divide the expression into two sub expressions*/
          Qd=""; Qd += the first separator
         for each node c in CC do
           begin
              divide c into a set of sub expressions EXP;
              for each con in EXP do
                c1='['; c2=']'; qc1="";
               if con contains c1 then
                 begin
                   get the node node1 before c1;
                   if (node1 != NULL) then Qd += node1 + '[' + predicate; endif
                  find another character c2 matched with c1 ;
                   get string str1 between c1 and c2;
                  qc1 = Query_rewritten(str1, userLabel) ;
                end  /* if con contain c1 */
              else begin
                        divide con into a set of searching conditions SC
                        for each search condition qc in SC do  /*qc contained predication conjunction*/
                        if qc contain one of the operations <,<=, >, >=, = then
                            divide qc into q/*queried node*/, op/*operation*/), const/*constant*/, and conj/*conjuction, can be null*/;
                            qc1 += q + '[' + predicate + ']' + op + const + conj;
                         else qc1  += q + '[' + predicate + ']';
                        endif
                     end
             if (con contains c1 and node1 !=NULL and qc1 != "") then  Qd += ' and ' + qc1 + ']'; else Qd += qc1; endif
             Qd += conjunction predicate after con
           end  /* for con*/
        Qd += separator '/' or '//' or other separators;
       end  /* for node c*/
       return Qd;
endQuery_rewritten
```

Fig 6. The pseudo code for the query rewritten

When a user submits a query request, the system checks every XML document involved in the query whether the document was created by the user or not. If it was created by the user, *Rule 11* is applied and the query results are returned without the MAC involvement.

If the document was not created by the user, the query processing continues. The label access policy is searched from the security repository to obtain the label access rules. Then the query request is parsed and the query is first rewritten according to the label of the user and read-access rule by adding the label of the user as a searching condition in the query. The pseudo code for the query rewritten is shown in Fig.6. Next, the rewritten query is executed, and the intermediate objects which do not satisfy the read-access rule will be filtered. While a query is processed in this way, an inference attack can be prevented, as illustrated in the following example 7.

Example 7. Assume that the label access rules, as shown in Fig.3, are applied. A user *u* with label *<unclassified, {Technique}>* in *Technical Department* submits a query: */companys/employee[salary =“6000”]/name*, and the label of object *(salary of Alice)* is *<secret, {Technique}>*. If the *salary of Alice* was not filtered out from the results of the query, then a user *u* can infer that the *salary of Alice* is *6000.* In order to prevent *u* from inferring unauthorized data by the query, the query above was rewritten by our system as: */companys[@label1<=1 and substring(@label2, 3, 1)=’1’]/employee[@label1<=1 and substring(@label2, 3, 1) = ’1’ and salary[@label1<=1 and substring(@label2, 3,1)=’1’] = ”6000”] / name [@label1 <=1 and substring(@label2, 3, 1)=’1’]* . When the rewritten query is executed, the object *salary of Alice* is filtered out and the results of the query do not contain *salary of Alice*.

5. Experimental Results

We tested our MAC model in an XML database system for a number of purposes: to demonstrate the validity of our MAC model; to process different types of queries; to evaluate the performance impact of adding our MAC into an XML database. The experimental results are presented in this section. The test environment was two desktop computers with 2.8GHZ CPU, 1G memory and Windows XP (SP2) operating system. The XMark benchmark dataset [26] was used in our experiments. We used the tools of XMark to generate XML documents of different sizes, assigned the labels to objects in XML schemas and documents, and measured the response time of different queries with and without the MAC be installed.

5.1 The labels for subjects and objects

We used two types of applications (*App1* and *App2*) in our experiments. The main feature of *App1* is that the larger the category for an object is, so a smaller number of users that can access it, and different from *App1*, the main feature of *App2* is that the larger the set of category for an object is, so a larger number of users can access it. The label types shown in Table 1 are used in the two applications. Two label access policies *AP1* and *AP2* were defined in the Table 2. From the label access policies in Table 2, we can see that the application *App1* is for a multilevel secure system and the application *App2* is not.

After the label types and label access policies were created, the administrator loaded the XML schemas and documents, and created four users *Lisa, Tom, Alice* and *Mary* for *App1* and two users *Mary, Tom* for *App2*. The administrator then assigned the labels shown in Table 3 to those users in these two applications. Next, the administrator assigned labels to the XML schemas and documents. Tables 4 and 5 illustrate the labels assigned to XML schema and documents.

Table 1. The label types

label component	Label component type	The set of elements
Level	Ordered component	Common<Private<Secret <Top secret
Category	Unordered component	Buyer, Seller, Maker

Table 2. The label access policies

The groups of access policies	Type of access	Label access policy
The first label access policy AP1 for App1	Read rule	The level of the subject is larger than the level of the object, and the category of the subject **contains** the category of the object
	Write rule	The level of the subject is equal to the level of the object, and the category of the subject is equal to the category of the object
The second label access policy AP2 for App2	Read rule	The level of the subject is larger than the level of the object, and the category of the subject **is contained in** the category of the object
	Write rule	The level of the subject is equal to the level of the object, and the category of the subject is equal to the category of the object

Table 3. The users in *App1* and *App2* applications

Name of users	Label assigned to the user
For App1	
Lisa	<Private, {Buyer, Seller}>
Tom	<Secret, {Buyer, Seller, Maker }>
Alice	<Common, {Buyer}>
Mary	<Secret, {Buyer}>
For App2	
Tom	<Secret, {Buyer, Seller, Maker }>
Mary	<Secret, {Buyer}>

Table 4. Label assignments for XML schema

For App1	
Objects in schema	Label
/site	<Common, {Buyer}>
/site/people/person/profile	<Private, {Buyer, Seller}>
For App2	
Objects in schema	Label
/site	<Common, {Buyer, Seller, Maker}>
/site/people/person/profile	<Private, {Buyer, Seller, Maker}

Table 5. Label assignments for XML documents

For App1	
Objects in document	Label
//site	<Common, {Buyer}>
//site/regions/asia/item	<Private, {Buyer, Seller}>
//site/people	<Private, {Buyer, Seller}>
//site/people/person/profile/age	<Secret, {Buyer, Seller, Maker}>
For App2	
//site/regions/asia/item	<Private, {Buyer, Seller}>

5.2. The queries and analysis of their results

The requests submitted by users in Table 3 are the following four queries:

Q1: //site/regions

Q2: //site/people/person/profile[/age= '33']

Q3: //site/regions/*/items/name

Q4: //site/open_auctions/open_auction/initial

Our tests for these queries over different sizes of documents are analogous, in the way that the main difference is that the numbers of nodes contained in the query results are increased when the sizes of the documents are increased. So instead of showing all the results we obtained - due space limit, we only show the results of *Q2* for *Tom* in an XML document with a size of 10.5MB after the document is labelled, which can be found at Appendix 1. Furthermore, in Table 6 we list the numbers of nodes returned for different applications, users and queries over XML documents before and after being labelled by our MAC model.

Table 6. Results of the queries in different applications

Applications	user	Query	Number of nodes returned with MAC	Number of nodes returned without MAC
App1	Alice	Q1	31407	34650
	Lisa		34650	34650
	Tom		34650	34650
	Mary		31407	34650
	Alice	Q2	0	77
	Lisa		0	77
	Tom		77	77
	Mary		0	77
	Alice	Q3	1185	1305
	Lisa		1305	1305
	Tom		1305	1305
	Mary		1185	1305
	Alice	Q4	720	720
	Lisa		720	720
	Tom		720	720
	Mary		720	720
App2	Mary	Q2	77	77
	Tom		77	77

For the application *App1*, based on Table 4, Table 5, *Rule 4*, *Rule 5* and *Rule 6*, all the nodes and sub-nodes of the path: *//site/regions* are labelled with <*Common, {Buyer}*>, except nodes and sub-nodes of the path: *//site/regions/asia/item.* The nodes and sub-nodes of the path:*//site/regions/asia/item* are labelled with <*Common, {Buyer, Seller}*>. According to the read-access rule of *AP1*, *Alice* could not access the nodes and sub-nodes of *//site/regions/asia/item* because her label is <*Common, {Buyer}*>. Therefore, the number of nodes for *Alice* returned from *Q1* is about 10% less than that of *Lisa and Tom*, and the number of nodes for *Alice* returned from *Q3* is also about 10% less than that of *Lisa and Tom.* Because the nodes and sub-nodes of the path:*//site/people/person/profile/age* are labelled with <*Secret, {Buyer, Seller, Maker}*>, these nodes are only obtained for the query requested by user *Tom* with the label <*Secret, {Buyer, Seller, Maker}*>. Since the label for *Mary* is <*Secret, {Buyer}*>, the label <*Secret, {Buyer}*> and <*Common, {Buyer}*> match the read-access rule in *AP1*. Therefore *Mary* can see all the nodes *Alice* can see. It is exactly what *App1* is designed for, and our model can support it effectively.

We can also use our system to support application *App2*. We only demonstrate the difference of the two applications for the query *Q2* because the results for other queries are analogous. According to *Rule 4*, *Rule 5* and *Rule 6*, the nodes of the path:*//site/people/person/profile/age* are labelled with <*Secret, {Buyer, Seller, Maker}*>. In application *App1*, *Mary* could not access the nodes labelled with <*Secret, {Buyer, Seller, Maker}*>, but in Application *App2*, she could see those nodes. This reflected the purpose of *App2*. It verified the fact that our system can support such an application which requires that "the larger the category for an object is, so a larger number of users that can access it". More other tests were conducted for various purposes (but due to space limitation, only these two cases are reported here). Based on our tests, we can summery that:

(1) Our MAC model is capable of defining different label access policies according to different application requirements. Tests on two types (*App1* and *App2*) of applications are reported in this paper. Our MAC model can support both of them. They demonstrate that our MAC model is effective and practical.

(2) Our approach provides tools for creating different policies based on different application requirements and produces the results users expected; it can assign labels to XML schemas and documents as well as users for different types of applications. It shows that our approach is very flexible.

5.3. The performance of the queries and analysis

For any operation in XML databases, the introduction of our MAC model would need more time to execute the query request: analyze and rewrite the query, and filter the intermediate results. Nevertheless, the extra time needed for the MAC should not cause a significant decrease in performance. The tests reported in this section are aimed to compare the response time of the system with and without MAC. The SELECT operation is the basis for update-involved operations. The cost difference between the read-only operation (SELECT) and update-involved operations can be calculated if the amount of data needed to be updated can be estimated. In most cases, such cost differences are far less than the costs of locating data (SELECT). For simplicity, we only tested SELECT operations in our experiments to measure the overhead of our MAC model. For queries *Q1* ~ *Q4*, we only show the response time differences for processing queries *Q1* over different sizes in our experiments, and because queries *Q1* ~ *Q4* are simple queries, the results are similar. In our experiments, the response time of the queries *Q1* ~ *Q4* are mainly determined by the sizes of the documents and are not much related to the user who submitted the query. *Q5*: //listitem//keyword, is a complex query which involves ancestor-descendant structural joins. Fig.7 and Fig.8 show the response time for processing queries *Q1* and *Q5* on documents of different sizes in the XML database with and without the MAC.

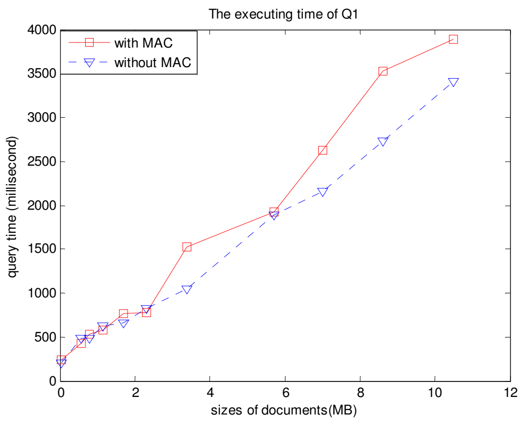

Fig.7 The executing time of Q1

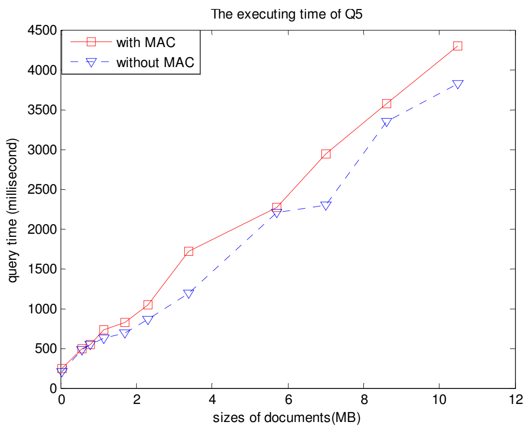

Fig.8 The executing time of Q5

Table 7 The sizes of the documents before and after the documents are labelled

Size of documents before labelled	0.027MB	0.56MB	1.15MB	2.3MB	5.7MB	7MB
Size of documents after labelled	0.041MB	0.8MB	1.7MB	3.4MB	8.6MB	10.5MB

From the figures (Fig. 7 and Fig. 8) and table (Table 7) above, we may observe the following facts:

(1) In our experiments, the sizes of XML documents before and after the documents were labelled are shown in Table 7. We can see that the size of documents increases by about 33% after the documents are labelled. The performance of the query process decreases due to the extra I/Os are required when the same queries are executed on the same documents after the documents are labelled (see Fig. 7 and Fig.8). When the sizes of documents are small, there is not much difference on response time for labelled and unlabelled documents, but when the size of documents increases the difference on response time can be clearly observed.

(2) When *Q1* and *Q5* are executed, one extra cost when MAC installed is filtering the intermediate results to obtain the final results (in addition to the cost of extra disk I/O access due to the documents being labelled and documents sizes have been increased). When the size of the documents is increased, the performance of the system decreases, but the processing time for the same query over the same document before and after the document being

labelled does not increase significantly, no matter the queries are simple or complex. As a result, it can be seen that the overall MAC system performance is rational, scalable and acceptable.

6 Related Work

A number of measures have been proposed for XML database security, including access control, encryption [9], secure XML information publishing [13], digital signatures and more, but it is generally agreed that access control is the most effective approach. There are usually three types of access control: mandatory access control (MAC), discretionary access control (DAC) and role-based access control (RBAC) [17, 25].

Based on the BLP model [2], SungRan Cho et al [4] proposed the first MAC model for XML documents. However, in the model [4] the structure of DTD needs to be modified, and the structure of XML documents also has to be changed. Dong-Zhan Zhang et al [23] presented an extended MAC model for XML. They used strategies analogous to E. Damiani's model [5] for authorizing privileges and resolving label assignment conflicts but they defined their own label assignment and overriding rules. Lan Li et al [15] proposed a MAC model for XML documents, in which labels can be assigned to users and elements/attributes in XML documents. When a user wants to access an object in an XML document, the label of the user and the label of the element/attribute are compared according to the *simple security property* and the *strict *-property* in the BLP model. Since these three models are all based on the BLP model, they inherited its drawbacks - lack of sufficient flexible mechanisms to define different security policies for different applications. Different from those models, our MAC model can define the label structures and label access policies according to the security requirements of XML applications. Not only we can define label access policies for XML database systems in the similar ways that models [4, 15, 23] can, but our model also can work for multi-purpose applications. Meanwhile constraints between a read-access rule and a write-access rule are maintained so that the integrity of data is preserved. In addition, the rules for update-related operations, for exceptional privileges of both ordinary users and the administrators are proposed to maintain the security of the XML documents.

For relational database systems, Walid Rjaibi [18] proposed a multi-purpose MAC policy to enhance the flexibility of the MAC model. The administrator can define label types, label access rules, and exceptional privileges. This policy is suitable for different requirements and applications in relational databases. In addition, Oracle 10g [14, 21] provides a label security product to enhance the flexibility of MAC. The models for Oracle databases are only suitable for relational databases, and in the model [18], the domination relationship between two labels is not discussed in detail. Inspired by the MAC policy in [18], a MAC model for XML databases is proposed. Meanwhile, the domination relationship in our model is not determined for any two labels because of the need of processing the *INTERSECTION* operation for label comparison (the reason was explained in *Example 4* in Section 2).

Another type of XML access control is based on DAC [3, 5, 8, 11, 12, 16, 24]. However, these models cannot resist attacks like *Trojan horse* that may result in information leaks. The reason was explained in *Example 6* (in Section 3).

XACML [17] is a specification of OASIS to express and deploy an access control policy based on XML and it is combined with RBAC models for XML [8, 25]. Although RBAC models are used in many applications, they can simplify the management of privileges of different roles for users. However, the limitation of the RBAC for XML has analogous security problems in DAC models because the invisible and sensitive data for a user or a role could be leaked by users or roles through transferring their own privileges to the other users or the roles (see Example 6). In contrast, our model can prevent such security problems which exist in DAC and RBAC models.

7. Conclusions

Providing mandatory access control model for databases is important for many database applications. In this study, we have developed a practical MAC model for XML databases for different purposes, including multilevel secure XML database systems. In our MAC model, label access policies can be defined according to the requirements of various applications, which enhance the flexibility of MAC models in general. A constraint between a read-access rule and a write-access rule has been proposed to maintain the integrity of data. Rules for label assignment and propagation are presented to alleviate the workload of label assignments, and a rule for solving label assignment conflicts is also introduced. Moreover, rules for modification-related operations, and exceptional privilege rules for ordinary users and the administrators are introduced. Nevertheless, the proposed MAC policy can be flexibly used in XML databases; therefore, it can be regarded as a generalization of BLP model for XML.

Our MAC model has been implemented in an XML database, and the experiments demonstrate the effectiveness of our approach; our approach can define different access policies, and is able to achieve and realize the design purposes of different access policies. A performance study on the overhead caused by introducing the MAC on an XML database was also carried out; the test results indicate that the performance is rational, scalable and acceptable overall.

In the future, we plan to extend out model to resolve the polyinstantiation problem for modification operations. We are also studying the issues on optimizing system performance.

Acknowledgements

The work presented in this paper is supported by 863 hi-tech research and development program of China, the granted number: 2006AA01Z430.

References

[1] V. Atluri, S. Jajodia, E. Bertino, Alternative correctness criteria for concurrent execution of transactions in multilevel secure databases, IEEE Transactions on Knowledge and Data Engineering, 1996, 8(5):839 – 854.

[2] D.E. Bell and L.J. LaPadula, “Secure Computer Systems: Unified Exposition and Multics Interpretation,” Technical Report MTR-2997, The Mitre Corp., Bedford, Mass., 1976.

[3] E. Bertino, E. Ferrari, Secure and Selective Dissemination of XML Documents, ACM Transactions on Information and System Security, 2002 , 5(3):290~331.

[4] S.R. Cho, S. Amer-Yahia, L. VS Lakshmanan, and D. Srivastava, Optimizing the Secure Evaluation Of Twig Queries. Int. Conference On VLDB, 2002.

[5] E. Damiani, V. SDC, S. Paraboschi, P.Samarati . A fine-grained access control system for XML documents. ACM Transactions on Information and System Security, 2002 , 5(2):169 ~ 202.

[6] Dom4J: the flexible XML framework for Java, http://www.dom4j.org/.

[7] Eclipse Downloads, http://www.eclipse.org/downloads/.

[8] W.F. Fan, C.Y. Chan, M. Garofalakis, Secure XML Querying with Security View. ACM SIGMOD 2004.

[9] J.Gao, T.J. Wang, D.Q. Yang, XFlat: Query-friendly encrypted XML view publishing, Information Sciences, 2008, 178, 774-787.

[10] H. He, R. Wong. A Role-Based Access Control Model For XML Repositories. in:Qing Li, Z.Meral Ozsoyoglu, Roland Wagner eds.. Proc of WISE 2000. Hong Kong,China. June, 2000. Hong Kong,China: IEEE Computer Society, 2000.138~145.

[11] M. Iwahara, R. Hayashi, S. Chatvichenchai, C. Anutariya and V. Wuwongse, Relevancy-Based Access Control and Its Evaluation on Versioned XML Documents, ACM Transactions on Information and System Security, Vol. 10, No. 1, Feb. 2007, Pages 1~31.

[12] Y. Kanza, A. O. Mendelzon, R.J. Miller, and Z. Zhang, Authorization-Transparent Access Control for XML under the Non-Truman Model, EDBT 2006.

[13] H.K. Ko, M.J. Kim, S.K. Lee, On the efficiency of secure XML broadcasting, Information Sciences, 2007, 5505-5521.

[14] J.E. Levinger. Oracle® Label Security Administrator's Guide10*g* Release 1 (10.1) Part No. B10774-01, December 2003.

[15] L. Li, X. Jiang, and J.H. Li, Enforce Mandatory Access Control Policy on XML Documents, in Proceedings in 7th International Conference of Information and Communications Security:, ICICS 2005, Beijing, China, December 10-13, 2005.

[16] M. Murata, A. Tozawa, M. Kudo,and S. Hada, XML Access Control Using Static Analysis, ACM Transactions on Information and System Security, Vol.9, No.3, Aug.2006, Pages 292–324.

[17] OASIS standard: eXtensible Access Control Markup Language (XACML) Version 1.0. http://www.oasis-open.org /committees /xacml/ repository /oasis-xacml-1.0.pdf, 18 February 2003.

[18] W. Rjaibi, P. Bird. A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

[19] XML Path Language (XPath) 2.0. W3C Recommendation 23 Jan. 2007, http://www.w3.org/TR/xpath20/.

[20] J.P. Yoon, Authorization: A Bitmap Indexing Scheme for High-Speed Access Control to XML Documents, IEEE Transactions on Knowledge and Data Engineering, Vol.18, No.7, July 2006, pages 971-987.

[21] X.D. Yuan, Y.Feng, The Model of Mandatory Access Control with Extended Security Label, Chinese Journal of Computers, October 2000: 1096~1100.

[22] T. Yu, D. Srivastava, L. V. Lakshmanan, and H. V. Jagadish. Compressed Accessibility Map: Efficient Access Control for XML. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 478–489, 2002.

[23] D.Z. Zhang, Y.S. Xue, An Extended Mandatory Access Control Model for XML, in Proceedings Advances in Computer Science – ASIAN 2005: 10th Asian Computing Science Conference, Kunming, China, December 7-9, 2005.

[24] H.X. Zhang, N. Zhang, K. Salem, D.H. Zhuo, Compact access control labeling for efficient secure XML query evaluation, Data & Knowledge Engineering, 2007, 60, 326-344.

[25] X.W. Zhang, J. Park and R. Sandhu, Schema based XML Security: RBAC Approach, Technical Report, IFIP WG 11.3, 2003.

[26] The XML benchmark project. Available from: <http://www.xml-benchmark.org>.

Appendix 1: The results of *Q2* for *Tom* over an XML document of 10.5MB (after the document is labelled). There are 77 nodes in the results.

```
<?xml version="1.0" key= "test6.xml"?>
<profile income="53373.22"><interest category="category55" />
<education>Other</education>
<gender>male</gender>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="23873.60"><gender>female</gender>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="47823.80"><interest category="category58" />
<interest category="category4" />
<interest category="category21" />
<education>Graduate School</education>
<business>No</business>
<age>33</age>
</profile>

<profile income="64448.45"><gender>female</gender>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="23092.52"><interest category="category9" />
<interest category="category10" />
<interest category="category26" />
```

```
<interest category="category51" />
<education>Graduate School</education>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="43900.78"><interest category="category56" />
<interest category="category57" />
<education>High School</education>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="65935.40"><interest category="category40" />
<interest category="category30" />
<gender>male</gender>
<business>No</business>
<age>33</age>
</profile>

<profile income="62004.90"><interest category="category37" />
<gender>male</gender>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="9876.00"><interest category="category30" />
<education>Graduate School</education>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="31709.85"><interest category="category50" />
<interest category="category46" />
<interest category="category31" />
<education>Other</education>
<business>No</business>
<age>33</age>
</profile>

<profile income="13560.28"><interest category="category29" />
<interest category="category3" />
<interest category="category40" />
<interest category="category29" />
<interest category="category27" />
<interest category="category18" />
<interest category="category10" />
<education>Other</education>
<business>Yes</business>
<age>33</age>
</profile>

<profile income="52458.60"><interest category="category40" />
<interest category="category20" />
<interest category="category16" />
<interest category="category44" />
<interest category="category48" />
<business>No</business>
<age>33</age>
</profile>
```