An Approach of Query Request Authorization Process for the Access Control System to XML Documents

Khandoker Asadul Islam **Yoshimichi Watanabe**,

Queensland University of Technology, Australia University of Yamanashi, Japan

Summary
Access control is one of the fundamental security mechanisms in information systems. When a multi-user system uses XML documents as data storage, the need of access control to XML documents arises. Due to the hierarchical structure, XML access control is fine-grained in nature. For this criterion, instead of controlling access to the whole XML document, it is possible to limit user access to substructures of the document. One of the key problems on which XML access control is centered is to find techniques for efficient enforcement of access control policy over XML data, thus user access authorization. In general, XML access control model uses XPath expressions for specifying the substructure of the document to define policy. Authorization process needs to find the substructure which is referring from the policy in order to evaluate user access to requested data. Thus, authorization process needs to access the data file every time user requests access to data. Evaluating concurrent requests on large data slow down the data access process especially on the Internet where large number of user accesses at any given time is very common. In this paper, we use classification of user requests and the user policy, and compare them to get the authorization result. Our experiment shows that the process significantly minimizes the need of data access in the process of evaluating user access.

Key words:
XML, Access control, Authorization

1. Introduction

A recent development in the database field has been the introduction of semi-structured and self-describing data, of which one example is data conforming to an XML format[15]. Such data collection can be referred as an XML database. XML is a promising standard for describing the structure of information and content on the Internet. The popularity of using XML as a data container is mostly because of its simplicity and richness of the data structure. W3C designated XML as the standard for Web data. For these special features, the use of XML databases in data communication and on the Internet is increasing in recent years. With the increasing number of applications that either use XML as their data model, or export relational data as XML documents, it becomes critical to investigate the problem of access control to XML documents.

Access control is one of the fundamental security mechanisms in information systems. It concerns with who can access which information under what circumstances. Typical implementations of access control are in the form of access control lists (ACLs) and capabilities. Using ACLs, a system maintains a list of subjects who have access to each object. In capabilities, each subject is associated with a list that indicates objects to which it has access. ACLs and capabilities are often not efficient for fine-grained access control over objects with complex structures, e.g. element-level access control in XML. XML access control is fine-grained in nature. Instead of controlling access to the whole XML document, it is often required to limit user access to the document (e.g. to some subtrees or to some individual elements). A number of standards such as XACL[9] and XACML[12] have already been proposed to address the issues of access control for XML documents. The other proposals and models concerned with this topic in [2], [4], and [10]–[12] are very useful.

The existing approaches deal with a number of different dimensions and offer different solutions. In general, XML access control policy is typically modeled as a set of access control rules. It is required to limit user access according to the policy. Most of the existing proposals of XML access control require to access XML data file while authorizing user requests. For example, the access control model proposed in Damiani's model[3] sets access rights to elements of XML documents and DTD files using DOM trees[16], and controls user's access to XML data according to the information of access rights in policy file which requires access to the data file. XPath expressions[18] are very popular to use to specify the substructure of the document on which the policy defines because it is a language from W3C for defining substructure of XML document. The substructure in policy is called the object of the policy. Only a few

Manuscript received February 5, 2010
Manuscript revised February 20, 2010

proposals including [11] and [13] work on authorizing user access without the use of actual data file. However, they do not efficiently consider the predicates in XPath expressions.

In this paper, we develop a technique for efficient enforcement of access control policy over XML data without the use of the actual XML database with few exceptions. We propose a simple and novel approach of checking authorization without run-time checking of XML data file. The exceptions are defined clearly and we describe the ways to working with those exceptions. We classify the XPath expressions based on the type of data it returns and significantly minimize the use of XML documents in authorization process by classifying the requested query and the object in policy in same way. Despite having some exceptions, our experiment shows that the overall performance is improved for cases with XML databases. XML databases are simple representation of commonly used 2-dimensional tabular form data which is commonly used in data communication and on the Internet for data storage.

The organization of the rest of this paper is as follows. Section 2 describes related works. Section 3 describes XML access control model and proposed classification for XPath expressions. Experiments are described in section 4, and in section 5 we evaluate our model with traditional access control models. Section 6 describes conclusion remarks, limitations and future works.

2. Related works

There are useful approaches for defining and enforcing access rights on XML documents. XACL[9] is an access control policy specification language based on 3-tuple (object, subject, action). XACL has flexible provisional authorization to a document based on whether there are certain conditions that the subject allows accesses to be logged and the subject signs an agreement. Bertino et al.[1] defines access rules at the schema level that apply to all documents conforming to the schema. They define to read elements or attributes, to modify/delete contents of elements or attributes, and to add/modify/delete elements or attributes.

Damiani et al.[3-4] specifies a language for encoding access restrictions. Rules in the specification language can be defined in DTDs/schemas or individual XML documents. In their approach, a rule essentially is 5-tuple (subject, object, action, sign, prop), allowing both negative rights (with the conflict resolution) and propagation to subtrees. Gabillon and Bruno[5] add numeric priority to resolve the conflict when multiple rules apply to an object. They implement access control by converting their authorization sheet to an XSLT document[19]. XACML[12] is an OASIS specification that is gaining acceptance for expressing access control policy for XML. XACML is based on the work including that of XACL, Damiani, and Bertino. It standardizes access request/response format and architecture of the policy enforcement framework; however it does not address deriving access control rules from the existing policy base.

Several authors have examined issues relevant to access control implementation. Jagadish et al.[8] present a space efficient accessibility map that identifies the user's accessible XML data items by exploiting structural locality of accesses in tree-structured data and a time efficient map lookup algorithm. Their work was not based on a specification of access rules but deciding accessibility given a set of access and conflict resolution rules. The Vimercati's authorization model for time-varying XML documents showed how one can pre-compute some rights when database contents are changed[14]. S. K. Goel et al.[6] showed that access rights derive data to be kept consistent with its sources in their approach.

M. Murata et al.[11] proposed a method of reducing burden of checking access control policies for XML documents by distributing the burden to static analysis and run-time check. Their key idea is to use static analysis and automata. The process generates automata from query, policy object and XML schema. Decision comes out by state transition of those automata. However, in some cases when XPath expressions in query and policy object are based on predicates, the model cannot statically make decision. They call it *value-based access control* and proposed over-estimated and under-estimated automata for these cases. However it is observed that in the process the procedure ignores the presence of predicates and makes decision while assuming expressions without it. In our model, the process does not need any schema and without it the effective analyzing method for the value-based access control is described. Also, access control system based on classification of query and object (our approach) is simpler than generating automata from query. That makes our approach simple and easily implementable effective access control model for XML databases.

N. Qi et al.[13] proposed a function-based model for providing expressive and scalable access control for XML databases. They presented two rule functions, ORF and SRF. High scalability was achieved by grouping rule functions into Java classes and further organizing classes into packages. Their proposed rule functions were based on a finite number of possible objects. In our approach, we consider XPath expressions on XML elements to define the object, thus considering every possible part of the document as permissible objects.

3. Access control to XML documents

Access control is a means to allow or deny subjects (user, process, roles, etc.) to operate (read, write, execute, etc.). When a user requests to access information or resources, the access control system decides whether the user can access them or not, according to the user's rights. XML document has a tree-typed hierarchical structure and is composed of several elements. An element corresponds to a node in the tree. For this structure, access control to parts of XML document is possible by assigning access rights to XML elements or nodes which represent a part of the XML document. Figure 1 shows the basic XML access control model.

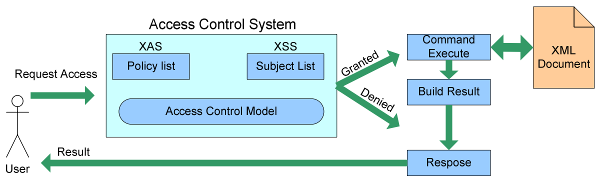

Fig. 1 A basic XML access control model

In the basic XML access control model, the system contains XAS (XML Access Sheet) which contains user rights and XSS (XML Subject Sheet) which contains user information. Each system follows a specific access control model. When the system gets an access request from a user, it authenticates the user with the help of XSS and authorizes the request according to the user's privileges in XAS. The user gets requested data or denial of access according to the authorization decision.

3.1 Access control policy

XML access control policy is typically modeled as a set of access control rules. In this paper, we assume the policy structure proposed in our previous paper[7]. Each rule is expressed as a 7-tuple (subject, object, action, type, propagation, administration, source), where (i) *subject* defines a set of subjects (e.g. user name, IP address, user role, or symbolic name); (ii) *object* defines a set of elements or attributes of the XML document (e.g. XPath expression); (iii) *action* denotes the actions that can be performed on the XML document (e.g. read, insert, update, delete, or all); (iv) *type* indicates whether this rule is a grant permission rule ("P") or a denial rule ("N"); (v) *propagation* refers to either local or recursive check (The value "R" stands for propagating the right to the subtree of the given node recursively, and the value "L" stands for not propagating to the subtree); (vi) *administration* represents whether the subject has the right to grant access to other user or not; and (vii) *source* holds the *subject* who assigns this permission.

For an example, the rule `(Alice, /CATALOG/CD, U+, P, R, Y, Bob)` says user "`Alice`" has a recursive update permission on "`/CATALOG/CD`" with administrative rights to grant permissions on "`/CATALOG/CD`" to other users, and the policy assigned by user "`Bob`".

3.2 Classification of objects

XPath expression in policy returns a part of XML documents on which a rule is defined. An XPath expression "`/CATALOG/CD`" returns entire "`CD`" elements under the element "`CATALOG`". In general, an XML database begins with a single element, called *root element*; in our example it is "`CATALOG`". Each element under this root element represents one data element. In this case, each "`CD`" element considers one data element. For simplicity of description, we will call such as "`/CATALOG/`" as level 1, such as "`/CATALOG/CD/`" as level 2 and such as "`/CATALOG/CD/TITLE`" as level 3 expressions. That is, the number of levels is given by the number of the character sequences divided by "/" in given XPath expression. For an example of XPath expression "`/CATALOG/CD[PRICE>10]/TITLE`" stands for the representation of level 3 with predicates which returns all "`TITLE`" elements under the "`/CATALOG/CD`" elements where the value of "`PRICE`" element is greater than 10.

In this paper, we classify XPath expressions used in objects of policy and in user query requests into four types. The types are (i) Type 1(Full), (ii) Type 2(Parts), (iii) Type 3(Elements) and (iv) Type 4(Conditional Elements). These are explaining below.

Type 1: This type of expression returns all the data elements. It is specified by the root element or other ways that returns all the data elements. Expressions of level 1 and 2 without predicates are Type 1 expressions. For example, "`/CATALOG`" and "`/CATALOG/CD`" belong to Type 1 expressions. As an exception, the expression "`/CATALOG/CD[PRICE>10]`" also belongs to this type when all "`PRICE`" element values are greater than 10.

Type 2: This type of expressions are represented by level 2 expressions with predicates when not all data element returns. The expressions "`/CATALOG/CD[PRICE>10]`" and "`/CATALOG/CD[TITLE="ABC"]`" are included in this type. For an example, the expression

"`/CATALOG/CD[PRICE>10]`" belongs to this type when there are some elements whose "`PRICE`" value is 10 or less.

Type 3: This represents by level 3 expressions with no predicates. For an example, the expression "`/CATALOG/CD/TITLE`" is included in this type.

Type 4: This is the representation of level 3 expressions with predicates. For example, the expression "`/CATALOG/CD[PRICE>10.8]/TITLE`" is a type 4 expression.

3.3 Checking the query request authorization

The authorization model assumes (i) closed world assumption; that is, if no rule applies to a request, the request is denied and (ii) denial rights override grant permissions as well as more explicit rules override less explicit ones. When a user submits requests, the authorization process authorizes the user request based on the policy. The authorization process can be divided into four separate steps shown in Fig. 2. Each step is explaining below.

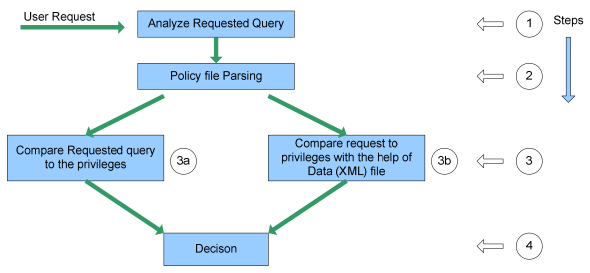
User Request
Analyze Requested Query
Policy file Parsing
Compare Requested query to the privileges
3a
Compare request to privileges with the help of Data (XML) file
3b
Decison
1
2
3
4
Steps

Fig. 2 Authorization process steps

The first step is to analyze the requested query and to determine the type. Second step is to parse the policy from policy file. Policy is expressed as an XML document; this step needs to parse the XML document. The process makes decision on third step. In this step, the requested query is matched with the user's rights. However, before the actual comparing, the determination of the type of object (XPath expression) in the policy is needed. After that only the types are compared without exception of Type 1. For example, if the type of the requested query is Type 1 then there should be a permission of Type 1 in the rights. In the same way, if the type of the requested query is Type 2 and that of the user has Type 1 permission, the request will be succeeded without exception of Type 1. For simplicity of describing the whole process, we consider only the grant permissions. However, in actual process denial permissions are evaluated first. By combination of requested query type and user's rights, either of the two comparisons (3a or 3b) is chosen, and the authentication process is completed.

We analyze the number of possible situations arose from comparing requested query type with the type of the objects. When we compared one type of requested query with any other type of right as a single possible case, we found that there could be a total of 16 possible cases for the four types of expression. The authorization process has a set of predefined comparison rules. Authorization decision has been made using these comparison rules. Figure 3 shows a set of possible comparison rules. However, there could be some variations when comparing expression types for XPath expressions with predicates. For example, if the user have grand permission on "`/CATALOG/CD[PRICE>10]`" and requested "`/CATALOG/CD[TITLE="ABC"]`", we cannot make decision by comparing these two expression types, although both expressions are belonging to Type 2. In these special cases, the procedure has to follow the 3b path in Fig. 2 and makes a run-time check to the XML document. However, if the user requested "`/CATALOG/CD[PRICE>12]`", that means if the field name of the predicates are same for both expressions, the procedure can make decision by evaluating a generated boolean expression. For predicates of policy expression and the predicate of user request "[*fieldname op1 value1*]" and "[*fieldname op2 value2*]"; the boolean expression, *"(fieldname op2 value2) implies (fieldname op1 value1)"* can decides whether user request is successful or not, where *op1* and *op2* are relational operators such as <, <=, >, and >=. For example, if the object of grant policy is "`/CATALOG/CD[PRICE>10]`" and user request expression is "`/CATALOG/CD[PRICE=12]`", then the decision making boolean expression would be "`(PRICE=12) implies (PRICE>10)`". The result is true so as the decision.

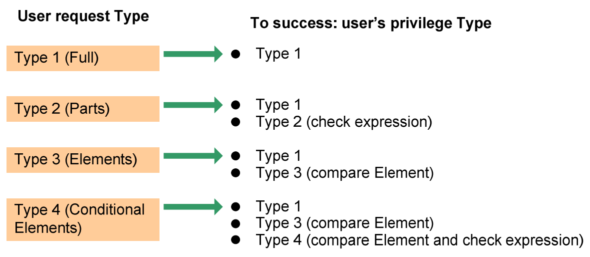

Fig. 3 Sample comparison rules

When we analyzed all the possible cases of comparison, we found that some Type 2 expressions may also return that of Type 1. For example, "`/CATALOG/CD[PRICE>12]`" expression looks like

Type 2, but it is belonging to Type 1 when "`PRICE`" are greater than 12 for all. We handle this situation as follows: when a Type 1 request fails against a Type 2 permission, the procedure reevaluate the Type 2 permission to check whether it is actually Type 1 or not. Only this time it requires accessing the data. We call it *confusing Type 2 expression.*

The authorization process would be very fast if we can follow the path through 3a in Fig. 2 for every possible case. This path does not use the actual data file, thus the processing time will be independent of the data file size. However, as mentioned special cases in the above, the procedure needs to follow the path through 3b. Although this step uses the data file to make decision, we can improve the performance with optimized implementation. For example, in our prototype we save the last comparison results and use it until the data file of the policy changes. Also we used serial access technique to access data when we only required doing that for the special cases.

4. Experiments

We built prototype software to test our authorization model. We generate synthetic policy data for randomly picked users from a fixed number of users. The synthetic policy generator considers every possible types of policy with every possible types of expression. In the experiments, every possible XPath expression is used as user query request, and the authorization process compares it with the different policies. The model has a tool which determines the expression type. We executed the algorithm on different XPath expressions, and we observed that it successfully classified the types. For the second step of the authorization process, we need to parse the policy file to search user policies. We collected the results of parsing time on different sizes of policy files. The result is shown in Table 1. Note that experiments are all based on XML databases with level 3 elements.

Table 1 Policy parsing time

No. of policy	Time (ms)
1,000	9
2,000	18
3,000	28
4,000	39
5,000	47
6,000	56
7,000	63
8,000	74
9,000	85
10,000	94

For measuring the authorization time, we set up the test on different sizes of data files. The test executed on a policy file containing 50,000 synthetic policy data. We observed the time for best cases (when it does not use data file) and for worst cases (when it checks the data file in run-time). Figure 4 shows the result. Type 1 & 3 for worst cases indicates the situation of confusing Type 2 expression. The result shown for type 2 & 4 is in worst case scenarios which means when we need any form of run-time data access. For evaluating the user request, the procedure has to parse the policy file, even though in cases when it does not check the data file in run-time. So the process always takes some time.

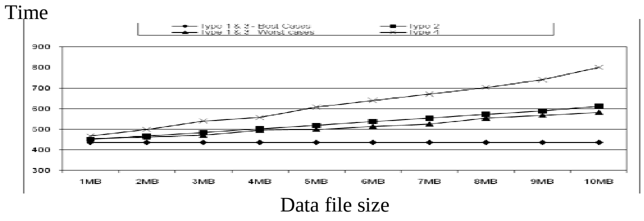

Fig. 4 Authorization time for different types of expression

The traditional process of XML access control accesses data file through building DOM trees[20]. When we tested our process against every possible query request, we observed the time required to build DOM tree on those cases separately. We compared only the DOM tree creation time with the authorization time in our method. Since the step of policy file parsing is common for both cases, we measured time without policy file parsing time. The result is shown in Fig. 5. Worst cases indicates the special cases when run-time database checking is required, and best cases means when the procedure can decide only by comparing expression types.

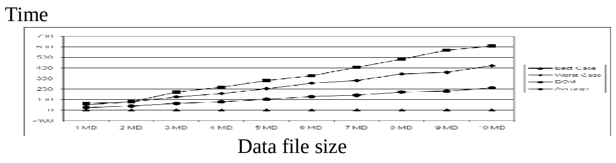

Fig. 5 Compare our process with DOM creation time

In general, the authorization processing time is independent of the data file size because it does not need to use the data file in the process. The process, however, needs to use the data file and the authorization processing time depends on the data file only for few special cases explained above. On those special cases, the process needs to evaluate the expressions by run-time checking the data file. The result from our prototype software indicates that proper optimized implementation accesses the data only when it absolutely needed. This improves the overall performance. In Fig. 5, the experiment result shows that the performance is better than creating DOM tree on that data file even if the cases are special. In our approach, the authorization process is faster than traditional processes even for the special cases.

5. Evaluation

We have already mentioned four separate steps in the authorization process. For the first step, the total execution time depends on the algorithm used for the classification of the expression. From our experiments, we saw that it was very negligible. The execution time for the second step, that is policy file parsing time, depends on the size of the policy file. This step is required for all other existing models. There are two possible ways for the third step. When it does not need to use the data file, the execution time depends only on the matching algorithm. It was very negligible according to the experiment results. The other way needs run-time checking of the data file. The execution time depends on the data file size. However, since we considered optimized implementation and serial access to data file for run-time checking in the prototype, we observed that the overall performance was improved.

In general, there are four steps in the traditional process of authorization checking, they are (i) parsing the policy file to get the user's rights, (ii) creating the DOM tree of the XML data file, (iii) creating the DOM tree with only the permitted nodes for that user, and (iv) executing the request on the new DOM tree.

In a quick glance, we make a summary of the execution time for the tradition process. The execution time of step 1 depends on the policy file size. That of step 2 depends on the size of XML data file. Step 3 depends on the algorithm to create the new DOM tree and its execution time depends on the size of data file. The execution time of step 4 also depends on the size of data file. By comparing these steps with our proposed steps, we found that in our process we could easily eliminate the time and space complexity for step 1 and 3 of the traditional procedure. We can also eliminate the time complexity of the step 4 except for few special cases. The authentication time is nearly constant for same policy file size and the authorization time is improved.

6. Concluding remarks and future works

In this paper, we are classifying requested query expressions and objects in policy. In our approach we showed how we could eliminate run-time data file checking in the process of authorization. We explained our process in four steps and showed experiment results for the proof of performance improvement. However, when any user has more than one policy, there would be a sub-step of combining all the policies before evaluating the request.

Our prototype software can successfully minimize the need of data file in the authorization process, but we identified some special matching cases where the process need run-time checking of the data file. In case of *confusing type 2 expressions*, we also need to run-time access to the data. The future work is to minimize these cases in the process of making authorization process fully independent of the data file. We considered only the XPath expressions as user request. The use of other query options such an XQuery [17] is in our future work plan. Because we considered only XML databases, the other of our future work will be to make the authorization process considering all forms of XML documents.

References

[1] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti: Specifying and Enforcing Access Control Policies for XML Document Sources, World Wide Web Journal, 3(3), 2000.

[2] E. Bertino and E. Ferrari: Secure and Selective Dissemination of XML Documents, ACM Transactions on Information and System Security, 2002, 290-331.

[3] E. Damiani, S. Vimercati, S. Paraboaschi and P. Samarati:Design and Implementation of an Access Processor for XML Documents, 9th International WWW Conference, 2000.

[4] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati: A Fine-Grained Access Control System for XML Documents, ACM Transactions on Information and System Security, 2002, 169-202.

[5] A. Gabillon and E. Bruno: Regulating Access to XML Documents, 15th Annual IFIPWG 11.3Working Conference on Database Security, 2001.

[6] S. K. Goel, et al.: Derived Access Control Specification for XML, ACM Workshop on XML security, 2003, 1-14.

[7] K. A. Islam and Y. Watanabe: Access Control to XML Document Including Administrative Authorization, 5th IASTED International Conference on Communications, Internet, and Information Technology, 2006.

[8] H. Jagadish, L. V. Lakshmanan, D. Srivastava, and T. Yu:Compressed Accessibility Map: Efficient Access Control for XML, International Conference on Very Large Databases, 2002.

[9] M. Kudo and S. Hada: XML Document Security Based on Provisional Authorization, ACM Computer and Communications Security, 2000, 87-96.

[10] C. H. Lim, S. Park, S. H. Son: Access Control of XML Documents Considering Update Operations, ACM Workshop on XML Security, 2003, 49-59.

[11] M. Murata, A. Tozawa, M, Kudo, S. Hada: XML Access Control Using Static Analysis, ACM Computer and Communications Security, 2003, 73-84.

[12] OASIS: Extensible Access Control Markup Language (XACML), http://www.oasis-open.org/committees/xacml/docs/, 2003.

[13] N. Qi, M. Kudo, J. Myllymaki, H. Pirahesh: A Function- Based Access Control Model for XML Databases, ACM 14th Conference on Information and Knowledge Management, 2005.

[14] S. D. C. di Vimercati: An Authorization Model for Temporal XML Documents, ACM Symposium on Applied Computing, 2002, 1088-1093.

[15] T. Bray, J. Paoli, and C. M. S. McQueen, Extensible markup language (XML) 1.0 W3C Recommendation, http://www.w3c.org/TR/REC-xml, 1998.

[16] A. L. Hors, P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, Document object model (DOM) level 3 core specification version 1.0, W3C Recommendation 07, 2004

[17] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon, XQuery 1.0: An XML query language, W3C working draft 16 august 2002. http://www.w3.org/TR/xquery/, 2002.

[18] J. Clarkand S. DeRose, XML Path Language (XPath) version 1.0, W3C Recommendation, http://www.w3.org/TR/xpath, 1999.

[19] J. Clark. "XSL Transformations (XSLT) Version 1.0". World Wide Web Consortium (W3C). http://www.w3c.org/TR/xslt (November 1999).

[20] Sun-Moon Jo andWeon-Hee Yoo, An Efficient Authorization Mechanism for Secure XML Sources on theWeb, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol.10 No.5, pp.721-727, 2006.

Khandoker Asadul Islam was born in 1976. He received his B. Sc. degree in Electrical & Electronics Engineering from Bangladesh Institute of Technology and M. Sc. in Computer Science from IBAIS University in 1999 and 2005 respectively. He received his Ph.D. in Information System Engineering from University of Yamanashi, Japan in 2008. He worked as a computer professional in various organization including Bangladesh Ministry of Planning and Bangladesh Board of Revenue as IT Consultant. Currently, he is working as a research fellow in Queensland University of Technology, Australia. His research interest includes XML, access control, information security, intrusion detection, cryptography and distributed software development.

YoshimichiWATANABE was born in 1964. He received the B. S. and M. S. degrees in computer science from University of Yamanashi in 1986 and 1988 respectively and received D. S. degree in computer science from Tokyo Institute of Technology in 1995. He is presently Associate Professor of the Department of Computer Science and Media Engineering at University of Yamanashi. His research interests include software development environment and software quality. Dr. Watanabe is a member of IPSJ, JSSST, JSQC, ACM, and IEEE.