Authorization Model with Provisions and Obligations in XML

Suhee Kim*, Jongjin Park*
* Department of Computer Science, Hoseo University, Korea
Tel : +82-41-540-5709 Fax : +82-41-548-9667 E-mail: shkim@office.hoseo.ac.kr

Abstract:

With the growing acceptance of XML technologies, XML will be the most common tool for all data manipulation and data transmission. Meeting security requirements for privacy, confidentiality and integrity is essential in order to move business online and it is important for security to be integrated with XML solutions. Many policies require certain conditions to be satisfied and actions to be performed before or after a decision is made. Binary yes/no decision to an access request is not enough for many applications. These issues were addressed and formalized as provisions and obligations by Betti *et Al.* In this paper, we propose an authorization model with provisions and obligations in XML. We introduce a formal definition of authorization policy and the issues involving obligation discussed by Betti *et Al.* We use the formal model as a basis to develop an authorization model in XML. We develop DTDs in XML for main components such as authorization request, authorization policy and authorization decision. We plan to develop an authorization system using the model proposed.

Key Words: Access Control, XML, Provision, Obligation, Authorization, Security

1. INTRODUCTION

We believe that XML will be the most common tool for all data manipulation and data transmission in the future. In order to move business online, it is essential to meet security requirements for privacy, confidentiality and integrity. With the growing acceptance of XML technologies for documents and protocols, it is logical that security should be integrated with XML solutions.

Policies are widely used in many systems and applications. It has been recognized that yes/no decision to an access request is not enough for many applications. Many policies require certain conditions to be satisfied and actions to be performed before or after a decision is made. These issues were addressed and formalized with provisions and obligations[1]. Provisions are those conditions that need to be satisfied or actions that must be performed before a decision is rendered, while obligations are those conditions or actions that must be fulfilled by either the users or the system after the decision.

In this paper, we describe our authorization architecture with provisions and obligations. We introduce a formal definition of authorization policy model which is presented in [1]. Specially, we introduce the issues involving obligation in policy management using the loan application [1] used as an example. We develop document type definitions (DTDs) in XML for main components in authorization model with provisions and obligations with support provisions and obligations in XML. We develop policy rules in XML for the loan example.

The rest of the paper is organized as follows. Section 2 briefly introduce related works. In Section 3, we present our authorization architecture. Section 4 and Section 5 introduce a formal model for policy rules and apply the model for loan application. We develop DTDs in XML for main components in authorization model in Section 6, and conclude the paper in section 7 with future work.

2. RELATED WORKS

There are many papers on access control policies to model role-based access control, delegation, time-dependent access control, multiple access control policies and so forth[3,4,5]. All these models, however, assume that the system either authorizes the access request or denies it.

A provisional authorization model was introduced recently by Kudo and Hada[2]. They propose an access control system for XML documents where provisional actions are considered in the specification of authorizations. Provisional actions are represented as functions such as log, verify, encrypt, transform, write, create, and delete to extend the semantics for authorization policies. These actions are executed by the system based on decisions of authorization requests by the subjects. This model has been generalized by Jajodia *et al.* where formal provisional authorizations and rules are proposed[6].

A formal model for provisions and obligations in policy rule management was introduced by Bettini *et al.* [2]. It provides algorithms to automatically derive weakest provisions and obligations. This paper focuses on the monitoring and managements of obligations. To address provisions and obligations in policy rule management, they used an example of a loan application and management. It allows users to initiate a loan application process if they are already registered in the system. If they are not already registered, they are given an opportunity to register with the system by supplying the necessary information and, if this step is successful, they are given permission to proceed with the loan application process. Note that here the initiation of the loan application is not a statically assigned permission to the users. Users are given the permission to apply for loan as long as they satisfy some conditions; if they do not satisfy these conditions, they are given a chance to perform certain actions to satisfy them. Assume a loan application is approved. Then, the applicant will have access to the funds under the condition that the user agrees to pay off the loan according to a certain payment schedule

3. AUTHORIZATION ARCHITECTURE WITH PROVISIONS AND OBLIGATIONS

Fig. 1 shows an architecture of authorization system with provisions and obligation. The authorization system assumes the existence of an interface through which a client can connect to the system. The interface invokes authentication and role checking modules that verify the client's identity and role. We may use any existing authentication mechanisms[7, 8] and role specification/ verification methods [9] for the design and implementation of these modules.

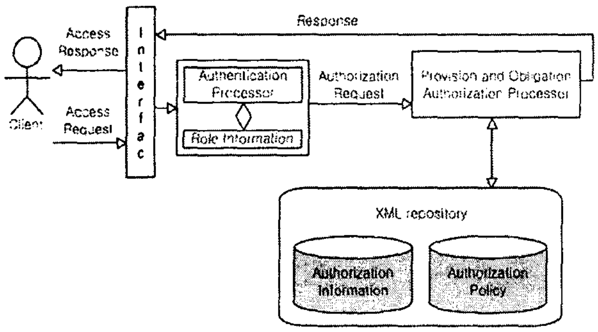


Fig. 1 Authorization Architecture with Provisions and Obligations

Once a client has been authenticated and the current role verified, the client may make access requests. An authorization request is passed to a provision-obligation authorization processor (POAP). Each authorization request involves permission to execute some action *a* on some object o. The POAP makes authorization decisions based on authorization policy rules and authorization information such as object and subject information.

4. FORMAL MODEL FOR POLICY RULES

Most studies in access control and authorization system deals with static policies where the results of user requests are only "Yes or No" decisions.

Here, we introduce a formal definition of authorization policy model which is presented in [1]. Specially, we introduce the issues involving obligation in policy management using the loan application as an example in [1]. Intuitively, a policy is represented as a set of basic rules where each rule is associated with a specification of its provisions and obligations. Let symbols **V**, **C**, and **Q** be finite sets of variables, constants, and predicate symbols, respectively. As in logic, atoms and rules are defined as follows:

- Atom: formula $Q(t_1 \ldots t_n)$ where Q is a predicate and each t_i is either a constant or a variable.
- Ground atom: atom with no variables
- Rule: formula $A \leftarrow B_1, \ldots, B_m$, $A, B_1, \ldots, B_m$: atoms, A : head, $B_1, \ldots, B_m$: body.
- Fact: rule with a head of ground atom and empty body

Let symbols P and O be two disjoint sets of provisions and obligations, respectively. These P and O sets are also disjoint from the set of predicate symbols Q. On the contrary, the set of variable and constant symbols V and C admitted in P and O predicates can used in the policy rules.

- Provisions: sets of predicate symbols P
- Obligations: sets of predicate symbols O
- Atom in provisions and obligations:
  a predicate $P_i(t_1, \ldots, t_k)$
  with P_i P or $O_i(t_1, \ldots, t_k)$ with O_i O
  and each t_i is either a constant or a variable.
- *PO*-atoms: atoms in provisions and obligations
- *PO*-formula: either *PO*-atom, or a disjunction or a conjunction of *PO*-formulas

An interpretation *I* of a *PO*-formula is a mapping from each ground atom to the constant True or False.

Table 1 shows a set of policy rules with provisions and obligations. The obligation $O_1(s, x, y)$ shows that an obligation may involve constant symbols (*s* in this case) that do not appear in the rules. A company may accept a deal imposing as obligation that the other party devolves part of the income to a charity organization. This fact can be identified by the constant *s* in O_1 and not to appear in any other policy rules.

Table 1 Set of rules with Provisions and Obligations(Ref.[1])

Policy Rule	PO-formula
$(R_1)\ Q_1(x) \leftarrow Q_2(x,y), Q_3(y)$	$O_1(s,x,y)$
$(R_2)\ Q_2(a, b) \leftarrow$	$P_1(b)$
$(R_3)\ Q_3(b) \leftarrow$	
$(R_4)\ Q_1(y) \leftarrow Q_4(z,y,c)$	$P_2(y,a) \wedge P_3(a) \wedge O_2(y,c)$
$(R_5)\ Q_4(c,a,c)$	

A policy decision is taken when a user satisfies a sufficient set of provisions and accepts the required obligations. Monitoring obligations accepted and taking appropriate actions upon fulfillment or defaulting are nontrivial tasks. For example, if the user agrees to pay a monthly fee for services as an obligation, the system needs to monitor this *obligation fulfillment*. In case of failure, the system may take compensating actions. Such compensating actions may decrease the *trustworthiness* of the user and replace *unfulfilled* obligations with (perhaps more costly) alternatives. In case of fulfilling obligations, a (positive) compensating action such as acknowledging payment of monthly fees, and upgrading to user's *trustworthiness* may be appropriate.

In order to specify the actions associated with fulfillment and defaulting we may add to an obligation expression a *fulfillment action specification* and a *defaulting action specification*, respectively. We may use the syntax shown in Fig. 2 to associate fulfillment and defaulting clauses to obligations.

```
OBL ::= [OBL Name
          Definition: obligationExpression
          FUL: ActionList
          DEF: <obligationExpression, ActionList >
   ]
ActionList ::= [Action List:{A1, ..., An}]
```

Fig. 2 Syntax of Obligation with Fulfilling and Defaulting Clauses in [1]

An optional name can be given to the obligation being defined as a convenience in rule specification. The FULfilling clause consists of a finite set of actions, to be invoked by the system at specified times, once the obligation has been fulfilled. The DEFaulting clause consists of a finite set of *compensating activities*. A compensating activity consists of an obligation instance and a finite set of actions. The semantics of a compensating activity is the replacement of the unfulfilled obligation with a (possibly empty) new obligation expression and a set of actions to be performed.
and the second is for the system itself to increase the reliability score of the customer.

5. POLICY RULES FOR LOAN APPLICATION

Fig. 3 specifies rules governing a loan application of a potential consumer buying an item for credit. Rule R1 states that any customer must satisfy the provision of registering with the vendor in order to read a loan application. Rule R2 states that any customer having read permission for a loan application can apply for the loan provided he signs a conditional purchase agreement with the vendor for a purchase and it in logged. Rule R3 states that a customer that has applied for a loan can get the loan *self approved* provided that the *reliability rating* of the customer is higher than 7.2, and has a monthly income of at least three times that of the monthly payment for the purchase. Furthermore, this approval obligates the consumer to accept the loan within 7 days of credit approval and, of course, obligates the consumer to pay back the loan. This is stated by the two obligations associated to R3. If the customer accepts these obligations, then the *system* will setup for the customer to sign up for the loan, and allow the purchase to proceed, as stated in the definition of the obligation buyWithinNumDays. If not, then the customer will receive a cancellation notice for the loan application. A customer accepting a loan, is obliged to make 36 equal payments of monthlyPayment on or before the 5th of every month, or make an additional payment of $100 on the 15th of every month. This is specified in the definition reported in Fig. 4 of obligation payLoan associated with rule R3.

These details are stated in the definitions of obligations payByDate and payByExtendedDate in Fig. 4.

Rule	PO-formula
access(customer, loan, read) ←	register(customer)
access(customer, loan, apply) ← access(customer, loan, read)	signConditionPurchaseAgreement(customer) $\wedge$ log (customer)
access(customer, loan, selfApprove) ← applicant(loan, customer), reliable(customer, score, time), (score≥7.2), computePay(customer, loan, monthlyPayment), monthlyIncome(customer, income), (income≥3*monthlyPayment)	log (customer) $\wedge$ buyWidthinNumDays(customer, loan, time, 7) $\wedge$ payLoan(customer, loan, monthlyPayment, 36)

Fig . 3 Policy Rules for Loan Application(Ref.[1])

```
OBL payLoan
Definition: [ for t = 1 to 36
[ if(¬(received(loanCanceled, customer, t', loan) and
(t'≤30t)))
payByDate(customer, 30t+5, monthlyPayment)
     or      payByExtendedDate(customer,      30t+15,
monthlyPayment+100)
     ]    ]

OBL buyWithinNumDays
Definition: buyWithinNumDays(customer, loan, time,
numDays)
FUL: [ Action List: { setUpSigning(customer, loan),
          proceedToSellItem(customer, time+numDays)
}]
DEF: [ Action List: { send(dealCancelNotice, system,
customer, timer+numDays, loan) } ]

OBL payByDate
     Definition: payByDate(customer, loan, time,
payment, penalty, upScore, downScore)
FUL: [ Action List: { send(acknowledgeReceipt, system,
customer, time, loan, payment),
send (adjustReliability, system, time, customer, upScore) } ]
DEF: [ OBL: payByExtendedDate(customer, time,
payment+penalty) ]
[ Action List: { send(reminder, system, customer, time,
loan, payment+penalty),
send(adjustReliability, system, system, time, customer, -
downScore) } ]

OBL payByExtendedDate
     Definition: payByExtendedDate(customer, time
payment, downScore)
FUL: [ Action List: { send(acknowledgeReceipt, customer
time, loan, payment) } ]
DEF: [ Action List: { send(cancelLoan, system, customer
time, loan),
send(adjustReliability, system, system, time, customer, -
downScore) } ]
```

Fig. 4 Obligation with Compensating Actions(Ref.[1])

6. REPRESENTAION OF AUTHORIZATION MODEL IN XML

In Sections 4 and 5, we have introduced a formal definition of authorization policy rules in [1] and the issues involving obligation in policy management using the loan application as an example. Main components in authorization system are authorization request, authorization policy and authorization decision. In this Section, we develop DTDs to represent these components in XML, and then apply DTD to specify policy rules for the loan example.

6.1 Authorization Request

Authorization request is a kind of query whether or not a customer can access an action on an object. It may consist of 3 components: Request (Subject, Object, Action). We simply specify this in a DTD in XML as in Fig.5.

```
<!ELEMENT Request (OBJECT, SUBJECT, ACTION)*>
<!ELEMENT OBJECT EMPTY>
<!ATTLIST OBJECT href CDATA #REQUIRED>
<!ELEMENT SUBJECT (UID,GROUPS?,ROLES?)>
<!ELEMENT UID (#PCDATA)>
<!ELEMENT GROUPS (GROUP)+>
<!ELEMENT GROUP (#PCDATA)>
<!ELEMENT ROLES (ROLE)+>
<!ELEMENT ROLE (#PCDATA)>
<!ELEMENT ACTION EMPTY>
<!ATTLIST ACTION name CDATA #REQUIRED>
<!ATTLIST ACTION name
(read|write|apply|selfApprove|log) #REQUIRED>
```

Fig.5 DTD of Authorization Request

6.2 Authorization Access Policy

Access policy discussed in the Section 4 consists of many rules. Each rule has head, body, provisions and obligations: Rule(head, body, provisions, obligations). We specify access policy DTD in Fig.6.

```
<!ELEMENT RULE (HEAD, BODY?, PROVISIONS*,
OBLIGATIONS*)>
<!ELEMENT HEAD (OBJECT, SUBJECT, ACTION)>
<!ELEMENT OBJECT EMPTY>
<!ELEMENT SUBJECT (UID,GROUPS?,ROLES?)>
<!ELEMENT UID (#PCDATA)>
<!ELEMENT GROUPS (GROUP)+>
<!ELEMENT GROUP (#PCDATA)>
<!ELEMENT ROLES (ROLE)+>
<!ELEMENT ROLE (#PCDATA)>
<!ELEMENT ACTION EMPTY>
<!ELEMENT BODY (ATOM)*>
<!ELEMENT PROVISIONS (PROVISION)*>
<!ELEMENT OBLIGATIONS (OBLIGATION)*>
<!ELEMENT ATOM (PARAMETER|ATOM)*>
<!ELEMENT PARAMETER (#PCDATA)>
<!ELEMENT PROVISION (ATOM)*>
<!ELEMENT OBLIGATION (ATOM*, FULFILL?,
DEFAULT?)>
<!ELEMENT FULFILL (ATOM)*>
<!ELEMENT DEFAULT (ATOM)*>
<!ATTLIST OBJECT href CDATA #REQUIRED>
<!ATTLIST ACTION name
(read|write|apply|selfApprove|log) #REQUIRED>
<!ATTLIST RULE name CDATA #REQUIRED>
<!ATTLIST ATOM name CDATA #REQUIRED>
<!ATTLIST ATOM value CDATA #IMPLIED>
<!ATTLIST PARAMETER name CDATA #REQUIRED>
```

Fig.6 DTD of Authorization Request

6.3 Authorization Decision

An authorization decision is a decision for a user's authorization request. It may consist of 6 components: Decision (Subject, Object, Action, Provisions, Obligations, Permission). We express this in a DTD in Fig.7.

```
<!ELEMENT Decision (OBJECT, SUBJECT, ACTION,
PROVISIONS*, OBLIGATIONS*, PERMISSION)>
<!ELEMENT OBJECT EMPTY>
<!ELEMENT SUBJECT (ROLES?)>
<!ELEMENT ROLES (ROLE)+>
<!ELEMENT ROLE (#PCDATA)>
<!ELEMENT ACTION EMPTY>
<!ELEMENT PROVISIONS (PROVISION+)>
<!ELEMENT PROVISION EMPTY>
<!ELEMENT OBLIGATIONS (OBLIGATION+)>
<!ELEMENT OBLIGATION EMPTY>
<!ELEMENT PERMISSION EMPTY>
<!ATTLIST OBJECT href CDATA #REQUIRED>
<!ATTLIST ACTION name
(read|write|apply|selfApprove|log) #REQUIRED>
<!ATTLIST PROVISION name CDATA #REQUIRED>
<!ATTLIST PROVISION valid (true|false) "true">
<!ATTLIST OBLIGATION name CDATA #REQUIRED>
<!ATTLIST OBLIGATION valid (true|false) "true">
<!ATTLIST PERMISSION value (grant|deny) "deny">
```

Fig.7 DTD of Authorization Decision

6.4 XML Specification of Policy Rules for Loan

Policy rules for loan application specified formally in Section 5 are converted into XML format using the DTD for policy rules.

```
<?xml version="1.0" encoding="euc-kr"?>
<!DOCTYPE Policy SYSTEM "policy.dtd">
<Policy>
<RULE name="R1">
<HEAD>
<OBJECT href="/loans/loan"/>
<SUBJECT><UID>707</UID>
<ROLES><ROLE>customer</ROLE></ROLES>
</SUBJECT>
<ACTION name="read"/>
</HEAD><BODY/
<PROVISIONS>
<PROVISION>
<ATOM name="register">
<PARAMETER
name="subject">customer</PARAMETER>
</ATOM>
</PROVISION>
</PROVISIONS>
</RULE>
<RULE name="R2">
<HEAD>
<OBJECT href="/loans/loan"/>
<SUBJECT><UID>707</UID>
<ROLES><ROLE>customer</ROLE></ROLES>
</SUBJECT>
<ACTION name="apply"/>
</HEAD>
<BODY><ATOM name="access">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER name="object">loan</PARAMETER>
```

```
<PARAMETER name="action">read</PARAMETER>
</ATOM></BODY>
<PROVISIONS><PROVISION>
<ATOM name="log">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER name="object">loan</PARAMETER>
<PARAMETER name="time">current time</PARAMETER>
<PARAMETER name="rule">R2</PARAMETER>
</ATOM>
</PROVISION>
<PROVISION>
<ATOM name="signConditionalPurchaseAgreement">
<PARAMETER
name="subject">customer</PARAMETER>
</ATOM>
</PROVISION></PROVISIONS>
</RULE>
<RULE name="R3">
<HEAD>
<OBJECT href="/loans/loan"/>
<SUBJECT><UID>707</UID>
<ROLES><ROLE>customer</ROLE></ROLES>
</SUBJECT>
<ACTION name="selfApprove"/>
</HEAD>
<BODY>
<ATOM name="applicant">
<PARAMETER name="object">loan</PARAMETER>
<PARAMETER
name="action">customer</PARAMETER>
</ATOM>
<ATOM name=">=">
<ATOM name="reliable">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER name="score">score</PARAMETER>
<PARAMETER name="time">time</PARAMETER>
</ATOM>
<ATOM name="value" value="7.2"/>
</ATOM>
<ATOM name=">=">
<ATOM name="monthlyIncome">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER
name="income">income</PARAMETER>
</ATOM>
<ATOM name="*">
<ATOM name="value" value="3"/>
<ATOM name="computPay">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER name="object">loan</PARAMETER>
<PARAMETER
name="monthlyPayment">monthlyPayment</PARAMETER>
</ATOM></ATOM></BODY>
<PROVISIONS><PROVISION>
<ATOM name="log">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER name="object">loan</PARAMETER>
<PARAMETER                name="time">current
time</PARAMETER>
<PARAMETER name="rule">R3</PARAMETER>
</ATOM>
</PROVISION></PROVISIONS>
<OBLIGATIONS><OBLIGATION>
<ATOM name="buyWithinNumDays">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER name="object">loan</PARAMETER>
<PARAMETER name="time">time</PARAMETER>
<PARAMETER name="limit">7</PARAMETER>
</ATOM>
</OBLIGATION>
<OBLIGATION>
<ATOM name="payLoan">
<PARAMETER
name="subject">customer</PARAMETER>
<PARAMETER name="object">loan</PARAMETER>
<PARAMETER
name="monthlyPayment">monthlyPayment</PARAMETER>
<PARAMETER name="partition">36</PARAMETER>
</ATOM>
</OBLIGATION></OBLIGATIONS>
</RULE>
</Policy>
```

Fig.8 Policy Rules in XML for Loan

6.5 Flow chart for Requesting a Loan

Fig.9 show a flow chart to request a loan using the rules in Fig.3.

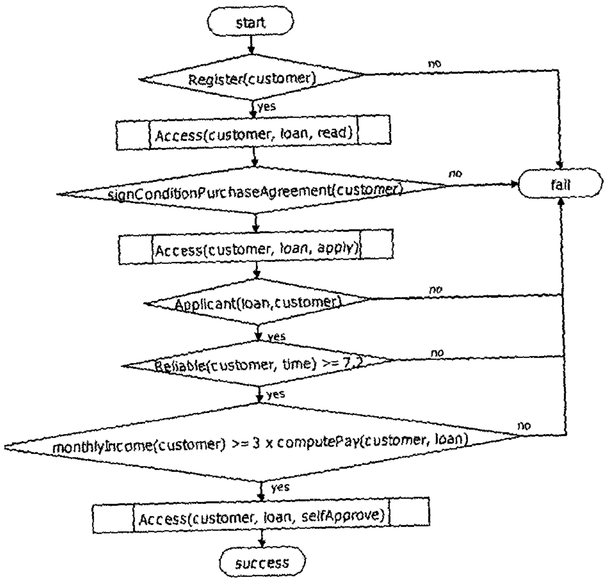
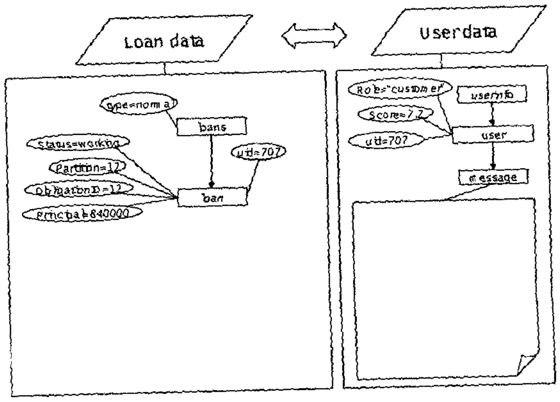


Fig.9 Procedure for Requesting a Loan

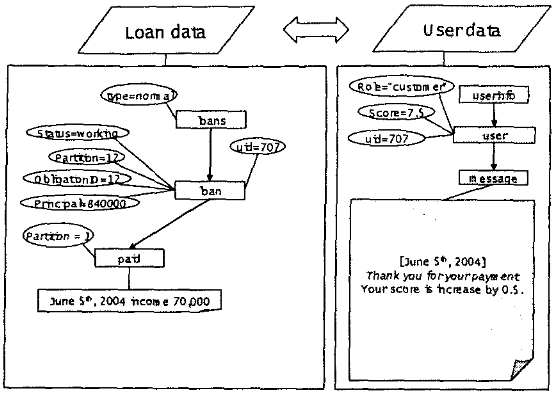
6.6 Monitoring a loan obligation

As we discussed in the previous sections, a loan s monitored after it is approved. This section illustrates state in each case.

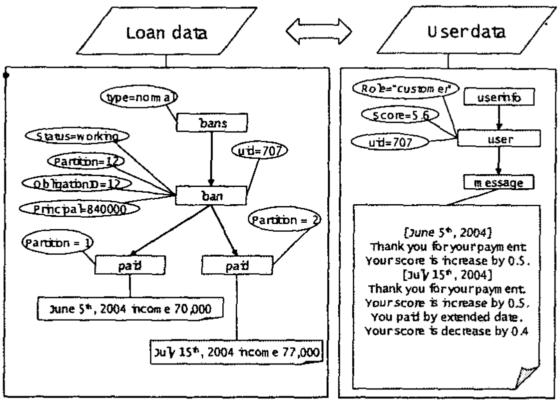
1) Request for a loan



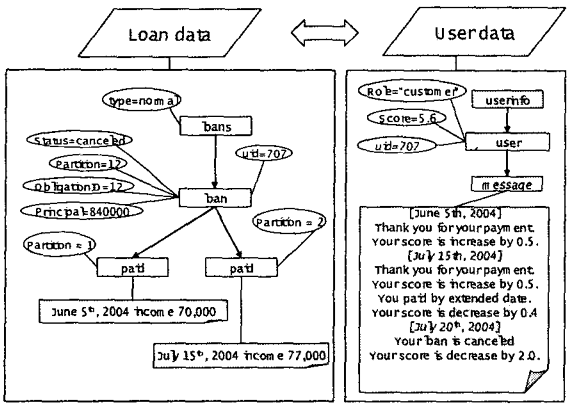
2) Fulfilling a payByDate Obligation



3) Fulfilling a payByExtendedDate Obligation



4) Cancel of the loan



7. CONCLUSION

In this paper, we proposed an authorization model with provisions and obligations in XML. We introduced a formal definition of authorization policy and the issues involving obligation addressed by Betti *et Al* using a loan example. We used the formal model as a basis to develop an authorization model in XML. We developed DTDs in XML for main components such as authorization request, authorization policy and authorization decision. We specified policy rules in XML for the loan application. We plan to develop an authorization system for an loan application using the model proposed.

References

[1] Claudio Bettini, Sushil Jajodia, X.Sean Wang, and Duminda Wijesekera. Provisions and Obligations in Policy Rule Management. *Journal of Network and Systems Management.* Vol. 11, No. 3, pp. 351-372 2003

[2] M. Kudo and S. Hada. XML document security based on provisional authorization. In *Proc. of the 7th ACM conference on Computer and communications security*, pp. 87-96, 2000.

[3] T. Y. C. Woo and S. S. Lam, Authorizations in distributed systems: A new approach, *Journal of Computer Security,* Vol. 2, Nos. 2/3, pp. 107-136, 1993.

[4] Elisa Bertino, Claudio Bettini, Elena Ferrari, and Pierangela Samarati, An access control model supporting periodicity constraints and temporal reasoning, *ACM Transactions on Database Systems*, Vol. 23, No. 3 pp. 231-285, 1998.

[5] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, andV. S. Subrahmanian, Flexible support for multiple access control policies, *ACM Transactions on Database Systems*, Vol. 26, (No. 2) pp. 214-260, 2001.

[6] Sushil Jajodia, MichiharuKudo, andV.S. Subrahmanian, Provisional authorizations. In Anup Gosh (ed.), *E-Commerce Security and Privacy*, Kluwer Academic Press, pp. 133-159, 2001.

[7] [KPS95] Charlie Kaufman, Radia Perlman, and Make Speciner. Network Security: Private Communication in a Public World. Prentice–Hall, Englewood Cliffs, NJ, 1995.

[8] [SNS88] J. G. Stener, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In Proc. USENIX Conf., February 1988.

[9] [BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In Proc. IEEE Symp. on Security and Privacy, pages 164–173, May 1996.